Delphi

简明教程

计 算 机 教 程

青苹果电子图书系列

Delphi 简明教程

郑　洪　　编著

张　晋　　审校

北京

Green Apple Data Center

前　言

Delphi 是由软件界巨头、著名的 Inprise 公司开发的“可视化”程序开发工具，它的基础语言为 Pascal 程序设计语言，同时它继承了 Pascal 语言的优点，如代码结构清晰、可读性好和代码执行效率高等，所以 Delphi 一经推出，迅速得到用户的好评。

经过 Inprise 公司的软件设计师们多年的改进，如今 Delphi 程序开发工具在各个方面都有了不同程度的提高，如数据库的体系结构、VCL 控件的开发、Web 应用程序和安全线程等，从而使得 Delphi 的功能更加的强大，使用起来也更加方便和灵活，提高了应用程序的开发效率。总之，Delphi 是开发 Windows 应用程序最为强大的工具之一，无论是 Windows 程序设计的初学者，还是有经验的 Windows 程序员，利用 Delphi 都可以迅速的开发出自己满意的应用程序。

为了满足广大读者的愿望，迅速地掌握 Windows 环境下的 Delphi 编程方法，编者以最新的 Delphi 5 为背景编写了本书，全书内容安排如下：

第一章介绍了利用 Delphi 编制应用程序的一些基础知识，以及如何利用 Delphi 编制简单的 Windows 应用程序；

第二章和第三章分别介绍了 Delphi 中对话框和菜单程序的设计；

第四章介绍了 Delphi 中常用的图形绘制和图像处理技巧；

第五章集中介绍了 Delphi 中各种窗体的开发技术，以及窗体的设计方案；

第六章介绍了 Delphi 中常用 VCL 控件（可视化控件）的开发应用程序，这是 Delphi 应用领域中非常重要的内容，也是读者应该必备的 Windows 编程本领；

第七章集中介绍了在 Delphi 中进行文件处理操作的方法；

第八章介绍了 Delphi 中非常吸引人数据库管理和开发方法；

最后一章（即第九章）介绍了多媒体程序开发的常用方法，在这一章中还设计了诸如声音播放器和视频播放器等实例程序，它将带领读者进入 Delphi 更高境界。

本书注重开发实例、开发经验、开发技巧和 Windows 高级特性开发，适合于各个层次的 Delphi 用户。对初学者来说，可迅速加入 Delphi 高级用户的行列；对有一定使用经验的读者，也可通过本书掌握 Delphi 深层次的开发方法，学会用更巧妙的办法开发出高水平的 Delphi 应用。然而，限于编者水平，书中的错误和不足之处在所难免，竭诚欢迎广大读者对本书提出批评和建议。

最后，书中示例中含有大量的程序代码、实例名称、图形图片、数据信息等，内容如有雷同，纯属巧合。

编者

Green Apple Data Center

内 容 提 要

Delphi 5 是由软件界的巨头人士、著名的 Inprise 公司开发的新一代可视化程序开发工具，一直以来受到计算机编程爱好者和各机构专业认识的广泛青睐。为了满足广大读者的愿望，迅速地掌握 Windows 环境下的 Delphi 编程方法，编者编写了本书。

全书共分为 9 章，主要包括 Delphi 5 中 VCL 控件的使用，文本、图形和图像的处理，菜单和对话框设计，文件管理技巧，多媒体程序制作，窗体设计方案，以及 Delphi 中引人入胜的数据库开发应用。书中融合了利用 Delphi 5 开发 Windows 应用程序的理论和实践，全面深入地介绍了利用 Delphi 5 开发应用的常用方法和技巧，实用性很强。

本书资料丰富、内容翔实，具有易学易用、简洁明了的特点，适合于对 Windows 编程有一定了解的读者使用，也可以作为广大计算机爱好者的参考读物。

Green Apple Data Center

目　　录

第一章　DELPHI 编程快速入门 1

1.1　Delphi 基本概念介绍 1
1.1.1　Delphi 的基本形式 1
1.1.2　面向对象编程的概念 1
1.2　Delphi 新特性简介 2
1.2.1　Object Pascal 语言功能扩展 2
1.2.2　改善 IDE 2
1.2.3　增强 VCL 控件 3
1.2.4　加快复杂工程的开发速度 3
1.3　系统配置和编程环境 3
1.3.1　Delphi 系统配置 3
1.3.2　Delphi 编程环境 4
1.4　第一个示例 6
1.5　小　结 11

第二章　WINDOWS 通用对话框 12

2.1　对话框函数简介 12
2.1.1　ShowMessage 函数 12
2.1.2　MessageDlg 函数 13
2.1.3　InputBox 函数 14
2.2　系统对话框调用技术 15
2.2.1　文件对话框 15
2.2.2　“颜色”对话框 17
2.2.3　“字体”对话框 19
2.2.4　“查找”对话框 20
2.3　标准对话框的使用 24
2.4　小　结 33

第三章　菜单程序的开发和设计 34

3.1　主菜单基本概述 34
3.1.1　创建菜单项 34
3.1.2　设置分隔条 37
3.1.3　设置快捷键 38
3.1.4　设置复选标志 40
3.1.5　菜单项有效状态设置 41
3.2　主菜单的应用 42
3.3　主菜单和弹出菜单综合应用 53
3.3.1　项目和设置 54

Green Apple Data Center

3.3.2 弹出菜单的设计.....55
3.3.3 运行结果和参考程序.....60
3.4 小 结.....63

第四章 DELPHI 中的图形绘制.....64

4.1 基本图形的绘制.....64
4.1.1 直线的绘制.....64
4.1.2 椭圆的绘制.....66
4.1.3 绘制多边形.....68
4.1.4 绘制矩形.....70
4.1.5 绘制三角曲线.....72
4.2 复杂图形的设计.....75
4.2.1 设计复杂图案.....75
4.2.2 实时绘画程序.....81
4.3 手工绘制图形.....86
4.4 小 结.....105

第五章 用户窗体的设计和使用.....106

5.1 快速使用向导.....106
5.2 多页窗体开发.....108
5.3 复合窗体开发.....118
5.3.1 绘制直线的子窗体设计.....118
5.3.2 绘制椭圆的子窗体设计.....119
5.3.3 绘制多边形的子窗体设计.....121
5.3.4 主窗体的完整设计.....123
5.4 多文档界面窗体的开发.....128
5.5 小 结.....137

第六章 常见 VCL 控件的使用.....139

6.1 Edit 控件和 Button 控件.....139
6.2 Listbox 控件和 Combobox 控件.....149
6.3 Memo 控件和 Dialog 控件.....152
6.4 Radiogroup 控件和 Checkbox 控件.....158
6.5 DirectoryOutline 控件和 Animate 控件.....168
6.6 小 结.....175

第七章 管理文件、目录和驱动器.....176

7.1 各种控件简介.....176
7.1.1 驱动器控件.....176
7.1.2 目录控件.....178
7.1.3 文件列表控件.....179
7.1.4 文件过滤器控件.....180
7.2 利用控件管理文件.....182
7.3 管理目录.....187
7.3.1 ChDir 函数.....187
7.3.2 MkDir 函数.....187
7.3.3 RmDir 函数.....188

Green Apple Data Center

7.3.4 应用实例 188
7.4 管理驱动器 193
7.4.1 DiskSize 函数 193
7.4.2 DiskFree 函数 194
7.4.3 GetDir 函数 195
7.5 小 结 196

第八章 数据库开发和管理 197

8.1 使用数据库向导 197
8.2 自己定制数据库 202
8.3 数据库管理程序 211
8.4 设计数据库报表 218
8.5 小 结 220

第九章 多媒体的设计和使用 221

9.1 认识 MediaPlayer 控件 221
9.1.1 AutoEnable 属性 222
9.1.2 Display 属性 223
9.1.3 Filename 属性 225
9.1.4 DeviceType 属性 226
9.1.5 EnabledButtons 属性 226
9.2 音频播放程序 227
9.3 视频播放程序 235
9.4 小 结 245

Green Apple Data Center

第一章　Delphi 编程快速入门

Delphi 是全新的可视化编程环境，为我们提供了一种方便、快捷的 Windows 应用程序开发工具。它使用了 Microsoft Windows 图形用户界面的许多先进特性和设计思想，采用了弹性可重复利用的完整的面向对象程序语言（Object-Oriented Language）和当今世界上最快的编辑器以及最为领先的数据库技术。

Delphi 使用面向对象的 Pascal（Orient Object Pascal 简称 OOP）作为基本开发语言。由于它继承了 OOP 语言中优越的全功能平台，弥补了可视化编程与全功能平台之间的障碍，以及编程效率高、代码质量好、面向对象等优点，解决了诸如 Visual Basic、Visual C++等可视化程序设计语言中的代码执行效率低下等问题。

对于广大的程序开发人员来讲，使用 Delphi 开发应用软件，无疑会大大地提高编程效率，而且随着应用的深入，您将会发现编程不再是枯燥无味的工作——“因为 Delphi 的每一个设计细节，都将带给您一份欣喜！”

1.1　Delphi 基本概念介绍

1.1.1　Delphi 的基本形式

Delphi 实际上是 Pascal 语言的一种版本，但它与传统的 Pascal 语言有天壤之别。一个 Delphi 程序首先是应用程序框架，而这一框架正是应用程序的“骨架”。在骨架上即使没有附着任何东西，仍可以严格地按照设计运行。您的工作只是在“骨架”中加入您的程序。缺省的应用程序是一个空白的窗体（Form），您可以运行它，结果得到一个空白的窗口。这个窗口具有 Windows 窗口的全部性质：可以被放大缩小、移动、最大最小化等，但您却没有编写一行程序。

因此，可以说应用程序框架通过提供所有应用程序共有的东西，为用户应用程序的开发打下了良好的基础。Delphi 已经为您做好了一切基础工作——程序框架就是一个已经完成的可运行应用程序，只是不处理任何事情。您所需要做的，只是在程序中加入完成您所需功能的代码而已。

在空白窗口的背后，应用程序的框架正在等待用户的输入。由于您并未告诉它接收到用户输入后作何反应，窗口除了响应 Windows 的基本操作（移动、缩放等）外，它只是接受用户的输入，然后再忽略。Delphi 把 Windows 编程的回调、句柄处理等繁复过程都放在一个不可见的 Romulam 覆盖物下面，这样您可以不为它们所困扰，轻松从容地对可视部件进行编程。

1.1.2　面向对象编程的概念

在此，请读者首先记住：面向对象的程序设计（Object-Oriented Programming，简记为 OOP）是 Delphi 诞生的基础。

OOP 立意于创建软件重用代码，具备更好地模拟现实世界环境的能力，这使它被公认为是自上而下编程的优胜者。它通过给程序中加入扩展语句，把函数“封装”进 Windows 编程所必需的“对象”中。面向对象的编程语言使得复杂的工作条理清晰、编写容易。说它是一场革命，不是对对象本身而言，而是对它们处理工作的能力而言。对象并不与传统程序设计和编程方法兼容，只是部分面向对象反而会使情形更糟。除非整个开发环境都是面向对象的，否则对象产生的好处还没有带来的麻烦多。而 Delphi 是完全面向对象的，这就使得 Delphi 成为一种触手可及的促进软件重用的开发工具，从而具有强大的吸引力。

一些早期的具有 OOP 性能的程序语言如 C++、Pascal、Smalltalk 等，虽然具有面向对象的特征，但不能轻松地画出可视化对象，与用户交互能力较差，程序员仍然要编写大量的代码。Delphi 的推出，填补了这项空白。您不必自己建立对象，只要在提供的程序框架中加入完成功能的代码，其余的都交给 Delphi 去做。欲生成漂亮的界面和结构良好的程序丝毫不必绞尽脑汁，Delphi 将帮助您轻松地完成。它允许在一个具有真正 OOP 扩展的

Green Apple Data Center

可视化编程环境中，使用它的 Object Pascal 语言。这种革命性的组合，使得可视化编程与面向对象的开发框架紧密地结合起来。

1.2 Delphi 新特性简介

到今天为止，Delphi 程序设计语言已经发展到了 5.0 版。和前期版本相比，Delphi 5 功能更加强大，使用更加方便。例如，Delphi 5 的集成开发环境界面换成了 Windows95 控制元件，并且对菜单的安排作了调整。可以建立的对象分类更为详细，许多部件是 Delphi 4.X 版所没有的；Delphi 5 中 DBE 5.0 的查询引擎经过重新编写后，不但功能增强了，速度也比以前的版本快了很多；Delphi 5 提供了 OLE Automation 功能，可以很方便地创建 OLE Automation Objec；Delphi 5 版还提供了新的存储各种资源的工具即所谓的“ Object Repository ”（对象仓储），用 Object Repository 可以存储所开发的表格及其它资源，更能体现软件重用的设计思想。

下面，我们就对最新版本的 Delphi 新特性作一比较系统的介绍：

1.2.1 Object Pascal 语言功能扩展

Delphi 5 扩展了 Object Pascal 语言的功能，增加了它的灵活性，具体包括以下几个方面：

（1）动态数组

在以前 Object Pascal 语言中，只能声明静态数组，也就是在声明数组的同时必须指定数组的长度。如果不能确定数组的长度，就必须把数组定义的尽量大，然而这就造成了内存资源的浪费。而使用动态数组，就可以事先不指明长度的情况下，让系统随着使用的情况自动的为其分配内存，提高了资源的使用效率。

（2）方法重载

在以前的 Object Pascal 语言中，一个方法对应唯一的一个方法，然而 Delphi 5 提供了方法重载的机制，即对象可以有一个或一个以上的同名方法。这些同名方法具有不同的参数类型和数目，因此编译器可以根据方法的参数类型和数目来确定使用了哪个方法。

（3）缺省参数

Delphi 的有些函数可能带有很多参数，在使用这类参数之前需要查阅函数说明，详细了解各个参数的使用方法和作用，而其中有些参数并不常用，往往是一些固定的值，因此 Delphi 5 提供了缺省参数功能，将这些不常用的参数设置为缺省参数，一方面保持了程序的灵活性，另一方面减少了程序员的工作量。

（4）64 位整数

Delphi 5 提供了 Int64 类型，它用于处理比 Longint 还要大的整数，表示数值的范围是-2^{63}~2^{63}-1。

1.2.2 改善 IDE

Delphi 5 中的集成开发环境（IDE）有了很大改进，能极大地提高开发效率，它主要依靠简化读写和浏览代码的操作来提高开发效率。

代码编辑器让浏览本单元或相关单元的内容变得容易，工程管理器中的拖放支持使得从已打开的工程中或资源管理器的对话框中选择文件并增加到工程的文件中变得更简单，使用工程管理器中多个工程管理的功能让用户能同时编辑多个工程的文件，应用 Delphi 5 的可视化窗口设计让用户能够轻松地从模板中选择创建 Internet 程序、分布式计算及 Windows 程序等等。

与 Delphi 4 相比，Delphi 5 的 IDE 新特点主要包括：

- 桌面的用户化设置增强了用户对开发环境的控制；
- 编辑键映射功能可以让编辑器按照用户的习惯来工作；
- 工程浏览可以帮助用户更好地理解代码，操作 VCL；
- 带有树视图和数据图表视图（Data Diagram View）的 DataModule 设计器可以帮助用户充分理解程序中的数据；
- 资源文件与工程管理器和编译器集成在一起；
- 动作列表保存了开发计划；

Green Apple Data Center

- 控制面板向导可以帮助用户定制应用程序的属性。

1.2.3 增强 VCL 控件

VCL（Visual Component Library）是 Delphi 实现代码重用的基本工具。在 Delphi 5 中，用户可以利用面向对象设计的强大功能开发出稳定、可靠、高效的程序，可以利用现有的面向对象的控件创建自己的控件。

Delphi 5 企业版的 VCL 中一共包含有 200 多个控件，利用这些控件，用户可以奇迹般地加快应用程序从开始开发到推向市场的速度。无论是开发 Windows 程序还是 Internet 应用，Delphi 5 基于控件的开发模式都能大大降低开发任务的难度。

Delphi 5 的 VCL 的新特点包括：

- 提供用于创建和重用复合控件的帧架构（Frames）；
- 增加了能将 HTML 浏览功能集成到应用程序的 WebBrowser 控件；
- 增加了 Microsoft Office 自动化组件集，以便把 Word、Excel 和 Outlook 等 Office 程序快速集成到应用程序中；
- 属性编辑器（Property Editors）支持自画（Owner Draw）功能，简化了属性选择；
- 增强了对高级自画（Advanced Custom Draw）函数的支持，以加强对 Windows API 的控制。

1.2.4 加快复杂工程的开发速度

Delph 5 增加了“模块窗口”。模块窗口能够显示当前所有映射到应用程序地址空间的模块，包括应用程序自身、应用程序显式或隐式调用的 DLL 以及操作系统调用的 DLL，模块窗口可帮助您优化程序结构。此外，Delphi 5 还能自动记录在调试过程遇到的事件，如断点、Windows 的消息以及其他调试信息。Delphi 5 能够监视指针错误，如果某个指针试图非法访问内存的某个地址，程序就会暂时中断运行，由调试器接管控制权，就好像遇到断点一样。

Delphi 5 进一步增强了调试功能，即使是非常复杂的工程，查找并修改错误都变得异常简单。Delphi 5 改进的调试器能帮助用户理解并控制自己编写的代码。用户利用断点可以在需要的地方深入代码进行调试，而现在，用户甚至可以在断点处设置触发断点后要采取的动作，并把断点进行分组集中在一起，以便快速调试自己感兴趣的代码段。

Delphi 5 的代码调试器包含以下一些新特点：

- 各种调试窗口和编辑器之间紧密集成，拥有一致的用户界面，支持剪贴板和拖动操作；
- 增加了断点提示条（Breakpoint ToolTips），以加快浏览和定位源代码；
- 新增的断点动作（Breakpoint Actions）功能加强了对调试进程的控制；
- 断点集合（Breakpoint Groups）方便了对多条断点的控制；
- 在 DCUs 和调试符号查找路径的帮助下，调试工作变得更简单；
- FPU 调试窗口支持 MMX 指令，从而增强了底层调试的能力；
- 进行多进程调试时，增加了临时进程选项，并支持调试子进程；
- 支持跨越进程边界的调试；
- 能够挂接到正在运行的进程上进行调用。
- Delphi 5 程序调试器支持远程调试、多线程调试，并且可以观察 CPU 调试窗口。

上面提到的种种新特性引人入胜，使得 Delphi 5 成为目前在 32 位 Windows 环境下最具有吸引力的开发工具。然而，Delphi 这个神圣殿堂到底会发出多么耀眼的光芒呢？且让我们拭目以待。

1.3 系统配置和编程环境

1.3.1 Delphi 系统配置

值得注意的是，要想在计算机上安装 Delphi 5，必须在计算机上先安装相应的硬件和软件系统，这些系统

Green Apple Data Center

要求包括：

- Windows 98/2000、Windows NT 4 **或更新版本（例如** Windows NT 4.X**）；**
- 80586 **或更高微处理器；**
- **一个** CD-ROM **驱动器；**
- Microsoft Windows **支持的** VGA **或分辨率更高的监视器；**
- **至少** 32 MB **内存（**RAM**），最好使用** 64MB **以上；**
- **完全安装大约需要** 180 **兆以上的硬盘剩余空间。**

1.3.2 Delphi 编程环境

Delphi **启动后，在它的缺省画面（软件开发界面）中，包括有主窗口、对象查看器、窗体窗口和代码窗口等，如图** 1-1 **所示。**

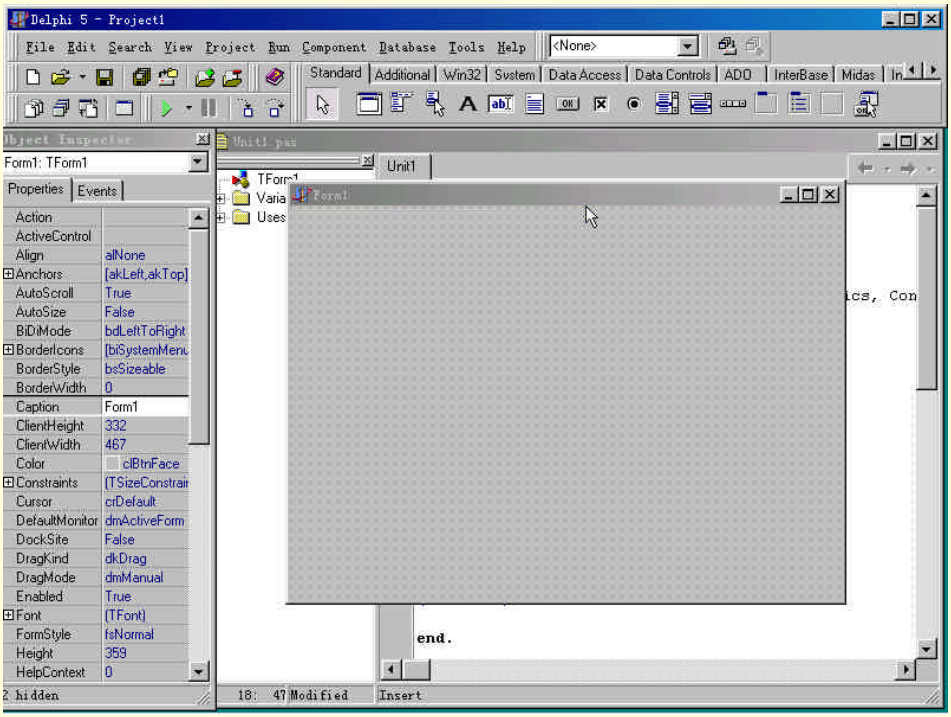


图 1-1 Delphi **开发界面**

提示：

> **为避免隐藏在** Delphi **后的** Program Manager **和曾经运行过的其它程序扰乱版面，分散您的注意力，不妨在启动** Delphi **前关掉其它应用程序；启动** Delphi **后，再最小化隐藏在后面的** Delphi 5 **程序组。这样屏幕上就只留下** Delphi **窗口可见了。**

首次加载 Delphi**，屏幕上会出现四个窗口：**

- **标题为“** Delphi-Project1 **”的** Delphi **主窗口；**
- Object Inspector **窗口；**
- **标题为“** Form1 **”的窗体（**Form**）窗口；**
- **标题为“** Unit1.PAS **”的代码编辑窗口。刚启动时这一窗口的大部分被“** Form1 **”窗体所掩盖。将“** Form1 **”窗体移开，或单击** Form1 **窗体下方的状态行，可以使其全部可见。在“** Form1 **”窗体的任意可见位置单击鼠标，可以恢复主窗体可见。**

1. 主窗口

Delphi **的主窗口位于屏幕的上端，包括** Menu**（菜单）、**Speed Bar**（加速条）和** Component Panel**（部件选项板）：**

- Menu **是下拉式主菜单。**

Green Apple Data Center

- Speed Bar 位于主窗口的左下端，由两排共 14 个加速按钮组成。这些按钮是菜单功能的快捷方式，各种图标直观地表示了它能执行的动作。
- Component Panel 由一行、若干页对象按钮所组成，利用它来选择需要的部件并将它放到窗体中去。

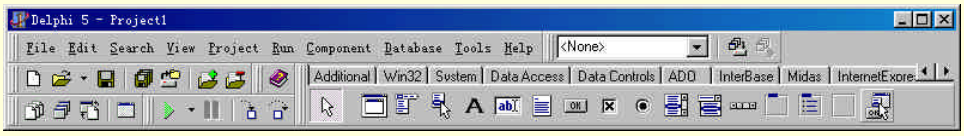


图 1-2 Delphi 主窗口

2. 对象检视器

Object Inspector 窗口含有两页：Properties 页显示窗体中当前被选择部件的属性信息，并允许改变对象的属性；Events 页列出了当前部件可以响应的事件。按动 Object Inspector 下端的“ Events ”页标签，使得 Events 页可见，这一定的事件后边的空白处，可以定义对象接受到相应事件时执行的动作。首次启动时，Object Inspector 窗口显示的是当前窗体 Form1 的属性。Object Inspector 根据对象属性的多少，决定是否有滚行显示。移动滚行条，可以查看当前对象的全部属性。

此外，Object Inspector 上还有 Object Selector（对象选择器），位于 Object Inspector 上方的下拉式菜单中。它显示了窗体上所有部件的名称和类型，也包含窗体本身。您可以用 Object Selector 很容易地在窗体的各个部件之间切换，也可以快速地回到窗体本身。当窗体中含有较多的对象时，您会发现这是切换对象尤其是回到窗体的最快捷途径。

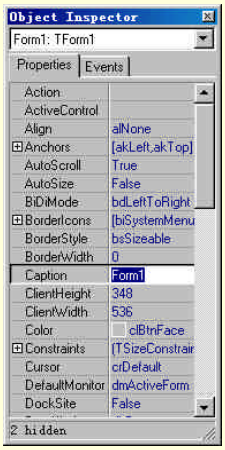


图 1-3 Delphi 对象检测器

提示：

> 想使 Object Inspector 一直可见，可将鼠标移到 Object Inspector 上，按动右键，以启动 Object Inspector 的弹出式菜单，将其设置为 Stay On Top。这对初学者常是一个很重要的设置方式。

3. 窗体窗口

Forms 窗口是开展大部分设计的工作区域。首次启动 Delphi 5 时显示的是窗体 Form1。可以把部件放在窗体中，通过移动位置、改变尺寸等操作随心所欲地安排它们，以此来开发应用程序的用户界面。您可以把窗体想象成一个可以放置其它部件的容器。窗体上有栅格（Grids），供放置部件时对齐位置用，在程序运行时 Grids 是不可见的。

Green Apple Data Center

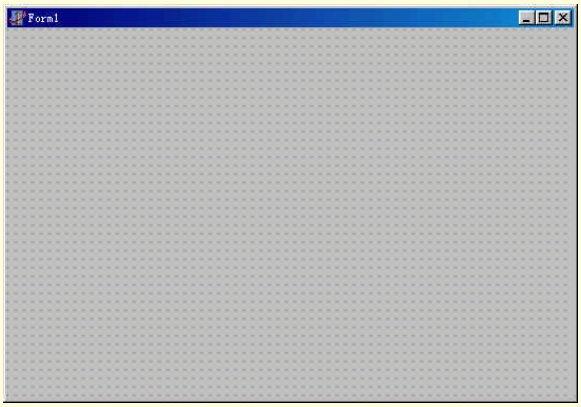


图 1-4 Delphi 窗体窗口

一个真正的应用程序可能有不止一个窗口，您可以选用不同的窗体进行设计。其它窗体可以是对话框（Dialog Box）、数据录入框等。

4. 代码窗口

代码窗口一开始处于窗体窗口之下。因为在 Delphi 中，设计用户界面直接在窗体中进行，运行结果和设计样板完全一致。当部件被放到窗体上时，Delphi 会自动生成大部分的用户界面代码。您所应做的只是在它为您生成的框架中加入完成所需功能的程序段而已。点动 Form1 的状态行使代码窗口可见。

如图 1-5 所示，显示了空窗体 Form1 的代码窗口。

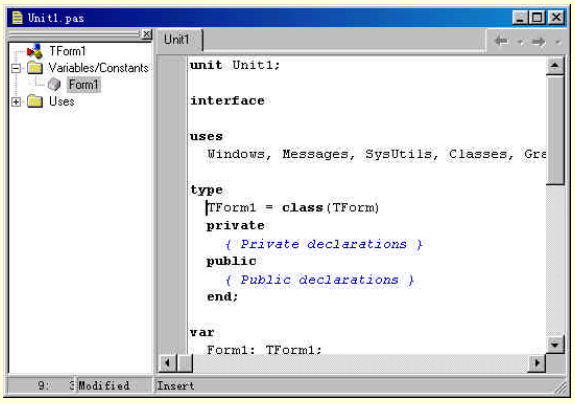


图 1-5 Delphi 代码窗口

1.4 第一个示例

下面我们用一个简单的示例程序，来说明在 Delphi 5 中进行应用程序开发的一般步骤和基本的方法，为以后的程序设计打基础。

编制示例程序的基本步骤如下所示：

1. 创建项目

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Standard 选项后，在 Memo 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 控件，接着向窗体上添加两个 Button 控件，添加控件后的窗体如图 1-6 所示。

Green Apple Data Center

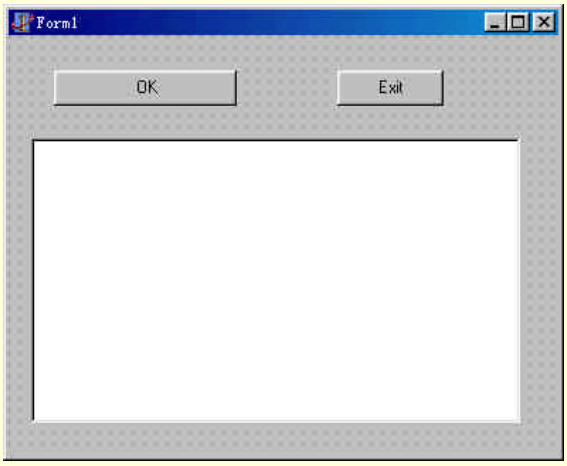


图 1-6　添加控件后的窗体

其中 Memo 控件用于显示文本，而 Button 控件用于显示一个对话框，窗体和控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 225
  Top = 107
  Width = 396
  Height = 323
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object Memo1: TMemo
    Left = 16
    Top = 72
    Width = 345
    Height = 201
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -24
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 0
  end
  object Button1: TButton
    Left = 32
    Top = 24
    Width = 129
```

Green Apple Data Center

```
    Height = 25
    Caption = ' OK'
    TabOrder = 1
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 232
    Top = 24
    Width = 75
    Height = 25
    Caption = 'Exit'
    TabOrder = 2
    OnClick = Button2Click
  end
end
```

2．程序初始化

程序的初始化过程，实际上就是对窗体 FormCreate()**事件的初始化，在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到** FormCreate()**事件的过程处理代码中，并且添加如下所示代码：**

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Button1.Caption:='Hello World';
//设置按钮标题
form1.Memo1.Clear;
//清除文本框
form1.Memo1.Lines.SetText('单击上面的 Hello World 按钮，您将会看到.....');
//从文本文件中读取信息
end;
```

在程序运行的初期，首先执行窗体 FormCreate()**事件中的代码，即通过语句** form1.Button1.Caption:='Click here';**来设置按钮控件的标题显示为字符串**'Click here'，form1.Memo1.Clear;**语句用来清空文本框中的显示内容，最后通过两条语句** form1.Memo1. Lines.SetText('**单击上面的** Hello World **按钮，您将会看到**.....')**设置一段文本信息，并且显示在文本控件** Memo1 **中。**

3．添加代码

在程序的设计过程中，用鼠标的左键双击窗体上的按钮控件，在屏幕上就会弹出一个代码窗口。在程序的设计阶段，把光标移动到代码窗口的 Button1Click()**事件处理过程中，并且添加如下所示的按钮响应代码：**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
showmessage('Hello World!');
{显示一个对话框}
end;
```

在程序运行的过程中，用户用鼠标的左键单击按钮时，那么程序就自动激活按钮控件的 Button1Click()**事件，然后通过语句** showmessage('Hello Wrold!')**来显示一个如图** 1-7 **所示的对话框。**



图 1-7　**显示对话框信息**

Green Apple Data Center

4. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，就会弹出一个如图 1-8 所示的保存文件对话框。

在弹出的对话框中选择合适的文件名保存文件。如图 1-8 所示，我们保存的单元文件为 hello.pas，之后保存项目文件为 project1.dpr。

保存程序文件后，按键盘上的组合键 Ctrl+F9 编译程序，然后再按 F9 键即可运行程序。在程序运行的初始画面中，首先按钮“OK”变为“Hello World”（请参看图 1-6 所示），同时文本控件显示一段幽默的文本信息，读者按照提示单击按钮控件，在窗体上就会弹出一个对话框，在其中显示“Hello World!”信息，程序运行的结果如图 1-9 所示。



图 1-8 保存文件对话框



图 1-9 程序运行结果

提示：

- 单击 Exit 按钮，即可退出此应用程序。

最后，附程序完整源代码如下所示，供读者参考学习 Delphi 应用的程序结构：

程序清单

```
unit hello;

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;


type
```

Green Apple Data Center

```
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Button1.Caption:='Hello World';
//设置按钮标题
form1.Memo1.Clear;
//清除文本框
form1.Memo1.Lines.SetText('单击上面的 Hello World 按钮，您将会看到.....');
//从文本文件中读取信息
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
showmessage('Hello World!');
{显示一个对话框}
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
//终止应用程序
close();
end;
end.
```

Green Apple Data Center

1.5 小 结

在本章中，列举了一个示例应用程序和一个向导演示程序，使得用户对 Delphi 5 可视化编程有一个感性的认识。通过本章的学习，读者可以对 Delphi 5 编程特性有一个大致的了解，带着问题和兴趣为以后的学习打下基础。

Green Apple Data Center

第二章 Windows 通用对话框

对话框是 Windows 操作系统中程序与用户沟通的一种常见交互方式，对话框可以向用户提供当前程序的运行状况，也可以接受用户输入的信息。Windows 对话框是 Delphi 可视化编程的特色之一，从某种意义上讲，也是衡量应用程序功能的一个方面，合理的利用 Windows 对话框界面，可以大大的提高应用程序的人机交互功能，增加用户与程序之间的交流机会。

在 Delphi 中，对话框的程序设计最常用的有两种方式——对话框函数标准标准对话框，在本章中，我们仍然以讲解结合示例的形式，对 Windows 对话框这种人机交互界面进行比较详细的讲解。

2.1 对话框函数简介

在 Delphi 中的对话框函数大体上可以分为两种——输入对话框函数和输出对话框函数，输入对话框函数用于接收用户在程序运行过程中输入的信息，其中包括 ShowMessage()函数、MessageDlg()函数，而输出对话框函数则用于显示一个对话框窗体，向用户报告当前程序的运行状态等信息，它包括 InputBox()函数下面就对各个函数分别的加以介绍。

2.1.1 ShowMessage 函数

对话框函数中的 ShowMessage()函数用于在程序运行的过程中显示一个包含一个字符串信息的对话框，用户只有对这个对话框进行正确的响应之后，才能够关闭对话框，进行下一步的工作。

ShowMessage()函数的语法结构如下所示：

```
ShowMessage(const Msg: string);
```

下面的示例程序就是用于在程序运行中产生一个对话框，在其中显示一个字符串，具体的步骤如下所示：

1．开始工作

首先其中一个新的项目，在屏幕上就会弹出一个空白的窗体，向窗体上添加一个按钮控件，控件的标题设置为“ Hello ”；

2．添加代码

在程序的设计过程中用鼠标的左键双击窗体上的按钮控件，在屏幕上就会弹出一个代码窗口，在其中添加如下所示的按钮响应代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var str:string;
//定义字符串变量
begin
str:='Welcome'+chr(10)+chr(13);
str:=str+'欢迎进入 Delphi 世界'+chr(10)+chr(13);
str:=str+'在这里将向您介绍 showmessage 函数的基本用法'+chr(10)+chr(13);
//为字符串变量赋值
showmessage(str);
//显示对话框
end;
```

Green Apple Data Center

3．运行程序

完成以上的工作后，选择菜单“File”中的“Save All”选项，在弹出的窗口中选择一个合适的文件名存储文件，然后按键盘上的功能键 F9 运行程序。

在程序的运行过程中，用鼠标的左键单击窗体中按钮“Hello”，在窗体上就会弹出一个对话框，如图 2-1 所示。

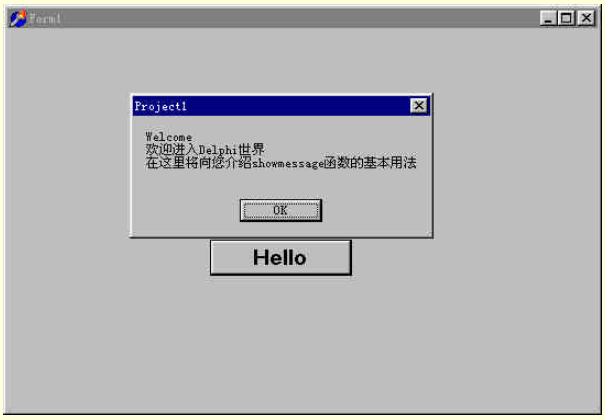


图 2-1　ShowMessage()函数输出对话框

2.1.2　MessageDlg 函数

对话框函数中的 MessageDlg()函数用于在程序运行的过程中显示一个包含一个字符串、位图和按钮信息的对话框，用户只有对这个对话框进行正确的响应之后，才能够关闭对话框，进行下一步的工作。

MessageDlg()函数的语法结构如下所示：

```
function MessageDlg(const Msg: string; AType: TMsgDlgType; AButtons: TMsgDlgButtons; HelpCtx: Longint): Word;
```

下面的示例程序就是用于在程序运行中产生一个对话框，在其中显示一个字符串，具体的步骤如下所示：

1．开始工作

首先其中一个新的项目，在屏幕上就会弹出一个空白的窗体，向窗体上添加一个按钮控件，控件的标题设置为“Welcome”；

2．添加代码

在程序的设计过程中用鼠标的左键双击窗体上的按钮控件，在屏幕上就会弹出一个代码窗口，在其中添加如下所示的按钮响应代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var str:string;
//定义一个字符串变量
begin
str:='Welcome'+chr(10)+chr(13);
str:=str+'欢迎进入 Delphi 世界'+chr(10)+chr(13);
str:=str+'在这里将向您介绍 showmessage 函数的基本用法'+chr(10)+chr(13);
//为字符串变量赋值
MessageDlg(str,mtInformation,mbYesNoCancel,0);
//显示一个对话框
end;
```

3．运行程序

在程序的运行过程中，用鼠标的左键单击窗体中按钮“Welcome”，在窗体上就会弹出一个对话框，如图 2-2 所示。

Green Apple Data Center



图 2-2　MessageDlg()函数输出对话框

2.1.3　InputBox 函数

对话框函数中的 InputBox()**函数用于在程序运行的过程中显示一个包含一个字符串和按钮信息的输入对话框，用户只有对这个对话框进行正确的响应之后，才能够关闭对话框，进行下一步的工作。**

它的语法结构如下所示：

```
function InputBox(const ACaption, APrompt, ADefault: string): string;
```

下面以一个示例来说明 InputBox()**函数的用法，具体的设计步骤如下：**

1．开始工作

在程序的设计阶段，向空白的窗体上放置一个按钮控件和一个输入文本框：

- **按钮空间** Button1**：用来显示一个数字输入框；**
- **文本框空间** Edit1**：用来显示信息输入框中输入数字的平方值。**

2．添加代码

并且在按钮的响应事件中添加如下代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var    str:string;
//定义字符串变量
begin
str:=inputbox('平方计算器','请输入原始数据：','');
//显示一个输入对话框
edit1.Text:=floattostr(strtofloat(str)*strtofloat(str));
//显示输入数据的平方值
end;
```

3．运行程序

存储文件，运行程序，在按钮控件上单击鼠标的左键，在窗体的上方就会出现一个如图 2-3 **所示的输入对话框。**



图 2-3　输入对话框

在输入对话框中输入一个正整数 234**，单击“** OK **”按钮，在程序运行的文本框中就会显示出** 234 **的平方值，如图** 2-4 **所示。**

Green Apple Data Center



图 2-4　程序运行结果

2.2　系统对话框调用技术

在 Delphi 5 中为了方便用户的开发工作，提供了十种系统对话框控件——打开文件对话框、存储文件对话框、打开图片对话框、存储图片对话框、打印对话框、打印设置对话框、字体设置对话框、颜色设置对话框、查找对话框和替换对话框。使用这些对话框，可以大大减轻程序开发的劳动量，所以读者一定要掌握这十种系统对话框控件。

2.2.1　文件对话框

通过调用 OpenDialog 控件和 SaveDialog 可以分别显示一个“打开”对话框和“另存为”对话框，如图 2-5 所示即为一个典型的“打开”对话框。

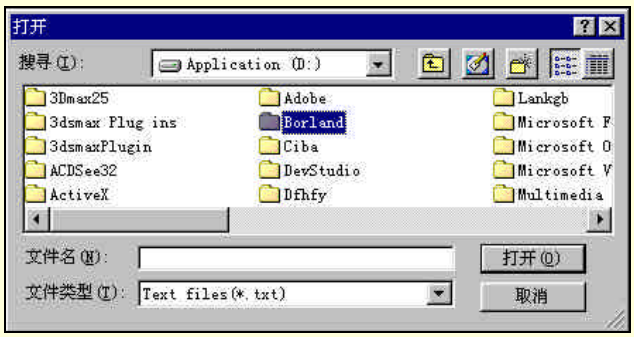


图 2-5　“打开”对话框

与“打开”和“另存为”这两个对话框有直接关系的属性有四个，它们是 InitialDir 属性、Filter 属性、FileName 属性和 Options 属性，它们的作用如下所示：

- IninDir 属性用于为“打开”或“另存为”对话框指定初始目录，如果此属性没有指定，则使用当前目录。
- Filter 属性用于返回或设置在对话框的类型列表框中所显示的过滤器。
- Filename 属性用于返回或设置所选文件的路径和文件名。
- Options 属性用于为“打开”和“另存为”对话框返回或设置选项。

其中 Options 属性对于“打开”和“另存为”对话框的特征设置有重要的作用，该属性是一系列设置值的集合，如表 2-1 所示为 Options 属性设置值及其说明。

Green Apple Data Center

表 2-1 “打开”与“另存为”对话框中 Options 属性设置值及其说明

常 数	常数描述
OfAllowMultiSelect	为真时，在文件列表框中可以对文件进行多重选择，通过按 Shift 键以及使用方向键可以选择多个文件，同时控件的 FileName 属性就返回一个包含全部所选文件名的字符串。
OfCreatePrompt	为真时，如果输入文件不存在时将会显示一个错误信息，并提示用户创建新文件。
ofExtensionDifferent	对话框所返回的文件扩展名与 DefaultExt 属性指定的扩展名不一致。
OfFileMustExist	为真时，用户在对话框中只能输入已经存在的文件名。如果文件不存在，就会显示一个警告信息。
OfShowHelp	为真时，在对话框中显示帮助按钮。
OfHideReadOnly	为真时，在对话框不显示只读复选框。
OfNoChangeDir	为真时，对话框打开时的目录设置为当前目录。
ofNoDereferenceLinks	为真时，不要间接引用快捷方式。
OfNoLongNames	无长文件名。
ofNoReadOnlyReturn	为真时，对话框返回的文件不能具有只读属性，也不能在写保护目录下面。
OfNoValidate	为真时，公共对话框返回的文件名中可以含有非法字符。
OfOverwritePrompt	为真时，使“另存为”对话框当选择的文件已经存在时应产生一个信息框，用户必须确认是否覆盖该文件。
OfPathMustExist	为真时，用户只能够在对话框中输入有效路径，如果输入非法路径时，会一个警告信息。
OfShareAware	为真时，忽略共享冲突错误。
OfNoNetworkButton	为真时，忽略网络驱动器。

下面以一个示例程序来说明各个参数对“打开”和“另存为”对话框的影响。

1.创建项目

首先启动一个新的项目，向窗体上添加一个按钮控件，并在控件工具栏上选择 Dialog 选项，向窗体上添加一个 OpenDialog 控件和一个 SaveDialog 控件。

2.添加代码

在按钮的响应事件中添加如下所示的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
form1.OpenDialog1.DefaultExt:='txt';
//设置缺省扩展名
form1.OpenDialog1.Title:='请选择一个文本文件:';
//设置对话框标题
form1.OpenDialog1.InitialDir:='d:\';
//设置缺省路径
form1.OpenDialog1.Filter:='Text files(*.txt)|*.txt';
//设置文件过滤条件
form1.OpenDialog1.Options:=[ofReadOnly, ofOverwritePrompt, ofNoChangeDir,
ofShowHelp, ofNoValidate, ofAllowMultiSelect, ofExtensionDifferent,
ofPathMustExist, ofFileMustExist, ofShareAware, ofNoReadOnlyReturn,
ofNoTestFileCreate, ofNoNetworkButton, ofNoLongNames, ofOldStyleDialog,
ofNoDereferenceLinks, ofEnableIncludeNotify, ofEnableSizing];
//设置对话框风格
form1.OpenDialog1. Execute;
//打开对话框
form1.SaveDialog1.DefaultExt:='txt';
```

Green Apple Data Center

```
//设置缺省扩展名
form1.SaveDialog1.Title:='请选择一个存储文件:';
//设置对话框标题
form1.SaveDialog1.InitialDir:='d:\';
//设置缺省路径
form1.SaveDialog1.Filter:='Text files(*.txt)|*.txt';
//设置文件过滤条件
form1.OpenDialog1.Options:=[ofReadOnly, ofOverwritePrompt, ofNoChangeDir,
  ofShowHelp, ofNoValidate, ofAllowMultiSelect, ofExtensionDifferent,
ofPathMustExist, ofFileMustExist, ofShareAware, ofNoReadOnlyReturn,
  ofNoTestFileCreate, ofNoNetworkButton, ofNoLongNames, ofOldStyleDialog,
  ofNoDereferenceLinks, ofEnableIncludeNotify, ofEnableSizing];
//设置对话框风格
form1.SaveDialog1. Execute;
//打开对话框
end;
```

程序说明：

在程序运行的初期，单击按钮时就会激活 procedure TForm1.Button1Click(Sender: TObject);**事件，其中的代码就会被执行。**

程序首先设置了“打开”对话框中允许对文件复选、对话框中包括“帮助”按钮并且对话框具有“资源管理器”风格，然后设置了对话框的标题为 "请选择一个文件："，**缺省工作路径为**"d:\"，**而且在对话框中将会显示所有文件，最后通过调用** Execute **方法来显示一个如图** 2-6 **所示的对话框。**



图 2-6　**选择打开文件的对话框**

在以上所打开的对话框中各个按钮的功能如下所示：



按钮：打开一个选中的文件；



按钮：取消所做的选择操作；



按钮：显示帮助信息。

“另存为”对话框的工作原理同“打开”对话框的工作原理大致相同，在这里就不在赘述了，请读者自己按照上述方法继续完成这个练习。

2.2.2　“颜色”对话框

通过调用 ColorDialog **控件的** Execute **方法可以显示一个“颜色”设置对话框，如图** 2-7 **所示即为一个典型的“颜色”对话框。**

Green Apple Data Center



图 2-7 “颜色”设置对话框

与“颜色”对话框有直接关系的属性只有一个 Options 属性，用于为“颜色”对话框返回或设置选项，同“打开”对话框一样，该属性也是一系列设置值的集合，如表 2-2 所示为 Options 属性设置值及其说明。

表 2-2 “颜色”对话框中常用的 Options 常数及其说明

常用常数	常数描述
cdFullOpen	为真时，显示全部的对话框，包括定义自定义颜色部分。
cdSolidColor	为真时，显示最相近的实体颜色。
cdPreventFullOpen	为真时，使定义自定义颜色命令按钮无效并禁止自定义颜色。
CdAnyColor	为真时，允许用户选用非实体颜色。
CdShowHelp	为真时，在对话框中显示帮助按钮。

下面以一个示例程序来说明 ColorDialog 控件属性设置对“颜色”对话框的影响。

1.创建项目

首先启动一个新的项目，向窗体上添加一个按钮控件，并在控件工具栏上选择 Dialog 选项，向窗体上添加一个 ColorDialog 控件。

2.添加代码

在按钮的响应事件中添加如下所示的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
form1.ColorDialog1.Color:=rgb(255,255,0);
//设置对话框初始颜色
form1.ColorDialog1.Options:=[cdFullOpen, cdPreventFullOpen, cdShowHelp, cdSolidColor, cdAnyColor];
//设置对话框风格
form1.ColorDialog1.Execute;
end;
```

程序说明：

在程序运行的初始阶段，单击按钮，就会激活按钮的响应事件，程序首先设置了在“颜色”对话框中可以包括“自定义颜色”按钮在内的所有颜色设置部分，而且可以在程序运行的过程中为“颜色”对话框设置初始颜色值。

如本示例程序通过语句 form1.Color Dialog1.Color:=rgb(255,255,0);设置了“颜色”对话框的初始颜色为黄色，最后通过 form1.ColorDialog1.Execute;语句来显示一个如图 2-8 所示的对话框。

Green Apple Data Center

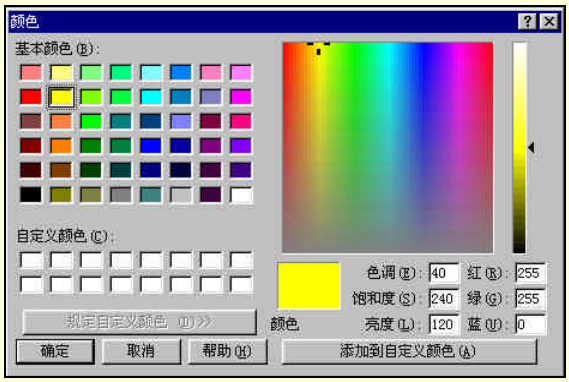


图 2-8 经过设置后的“颜色”对话框

2.2.3 “字体”对话框

通过调用 FontDialog 控件的 Execute 方法可以显示一个“字体”对话框，如图 2-9 所示即为一个典型的“字体”设置对话框。



图 2-9 “字体”设置对话框

与“字体”对话框有直接关系的属性有四个，它们是 Font 属性、MaxFontSize 属性、MinFontSize 属性和 Options 属性，它们的作用如下所示：

- Font 属性：指定在对话框中出现的最初字体，同时也可以指定用户返回的字体，通过这个属性还可以设置字体粗体、斜体等属性；
- MaxFontSize 属性：用于为“字体”对话框指定用户可以获得的最大字体尺寸；
- MinFontSize 属性：用于为“字体”对话框指定用户可以获得的最小字体尺寸；
- Options 属性：用于为“字体”对话框返回或设置选项。

其中 Options 属性对于“字体”对话框的特征设置有重要的作用，同“打开”对话框一样，该属性也是一系列设置值的集合，如表 2-3 所示为 Options 属性设置值及其说明。

表 2-3 “字体”对话框中常用的 Options 常数及其说明

常用常数	常数描述
fdAnsiOnly	为真时，对话框只允许选择 Windows 字符集的字体。如果该标志被设置，就不能选择仅含符号的字体。
fdApplyButton	为真时可以在对话框中显示“应用”按钮。
fdNoOEMFonts	为真时，在对话框中只有非矢量字体出现。
fdEffects	为真时对话框允许删除线，下划线，以及颜色效果。
fdFixedPitchOnly	为真时，可以指定对话框只能选择固定间距的字体。
fdForceFontExist	为真时，如果用户试图选择一个并不存在的字体或样式，显示错误信息框。
fdShowHelp	为真时，在对话框中显示帮助按钮。
fdLimitSize	为真时，对话框只能在由 Min 和 Max 属性规定的范围内选择字体大小。
fdNoFaceSel	为真时，在对话框中没有选择字体名称。
fdNoSimulations	为真时，对话框不允许图形设备接口 (GDI) 字体模拟。

Green Apple Data Center

fdNoSizeSel	为真时，对话框没有选择字体大小。
fdNoStyleSel	为真时，对话框没有选择样式。
fdNoVectorFonts	为真时，对话框不允许矢量字体选择。
fdScalableOnly	为真时，对话框只允许选择可缩放的字体。
fdTrueTypeOnly	为真时，对话框只允许选择 TrueType 型字体。
fdWysiwyg	为真时，对话框只允许选择在打印机和屏幕上均可用的字体。

下面以一个示例程序来说明 FontDialog 控件属性设置对“字体”对话框的影响。

首先启动一个新的项目，向窗体上添加一个按钮控件，并在控件工具栏上选择 Dialog 选项，向窗体上添加一个 FontDialog 控件，在按钮的响应事件中添加如下所示的代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
form1.FontDialog1.Font.Color:=rgb(255,255,0);
//设置字体颜色
form1.FontDialog1.Font.Size:=20;
//设置字体大小
form1.FontDialog1.Font.Style:=[fsBold, fsItalic, fsUnderline, fsStrikeOut];
//设置字体风格
form1.FontDialog1.Options:=[fdLimitSize];
//设置对话框风格
form1.FontDialog1.Device:=fdScreen;
//设置字体使用设备
if form1.FontDialog1.Execute then
    form1.Memo1.Font:=form1.FontDialog1.Font;
    //设置文本框的字体
end;
```

在程序运行的初始阶段，单击按钮，就会激活按钮的响应事件，程序首先设置了在“字体”对话框中初始化字体的大小、颜色和风格，并且设置了“字体”对话框的风格和使用设备，最后通过 form1.FontDialog1.Execute; 语句来显示一个如图 2-10 所示的对话框。

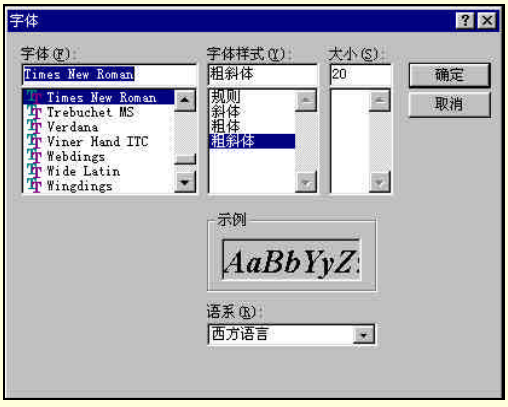


图 2-10 经过属性设置后的“字体”对话框

2.2.4 “查找”对话框

通过调用 FindDialog 控件的 Execute 方法可以显示一个“查找”对话框，如图 2-11 所示即为一个典型的“查找”设置对话框。

Green Apple Data Center



图 2-11 “查找”设置对话框

与“查找”对话框有直接关系的属性有两个，它们是 FindText 属性和 Options 属性，它们的作用如下所示：

- FindText 属性：存储欲查找的字符串；
- Options 属性：用于为“字体”对话框返回或设置选项。

其中 Options 属性对于“查找”对话框的特征设置有重要的作用，同“打开”对话框一样，该属性也是一系列设置值的集合，如表 2-4 所示为 Options 属性设置值及其说明。

表 2-4 “查找”对话框中常用的 Options 常数及其说明

常用常数	常数描述
frDisableMatchCase	为真时，大小写匹配复选框不可用。
frDisableUpDown	为真时，向上和向下两个按钮不可用。
frDisableWholeWord	为真时，全字匹配复选框不可用。
frDown	为真时，对话框中的向下按钮被选中，同时设置程序向下搜索字符串。
frFindNext	为真时，用户已经选中寻找下一个按钮。
frHideMatchCase	为真时，大小写匹配复选框不可见。
frHideWholeWord	为真时，全字复选框不可见。
frHideUpDown	为真时，隐藏向上和向下的按钮。
frMatchCase	为真时，大小写匹配复选框已经被检查过。
frReplace	为真时，一个系统设置的令程序用替换字符串去替换字符串的标志，仅对 ReplaceDialog 控件有效。
frReplaceAll	为真时，一个系统设置的令程序用替换字符串去替换所有查找字符串的标志，仅对 ReplaceDialog 控件有效。
frShowHelp	为真时，在对话框中显示一个帮助按钮。
FrWholeWord	为真时，全字匹配复选框可见。

下面以一个示例程序来说明 FindDialog 控件属性设置对“查找”对话框的影响。

1．创建项目

首先启动一个新的项目，向窗体上添加一个 Button 控件，并在控件工具栏上选择 Win32 选项，并且把其中的 RichEdit 控件添加到窗体上，然后选中控件工具栏上的 Dialog 选项，向窗体上添加一个 FindDialog 控件。添加完控件后的窗体如图 2-12 所示。

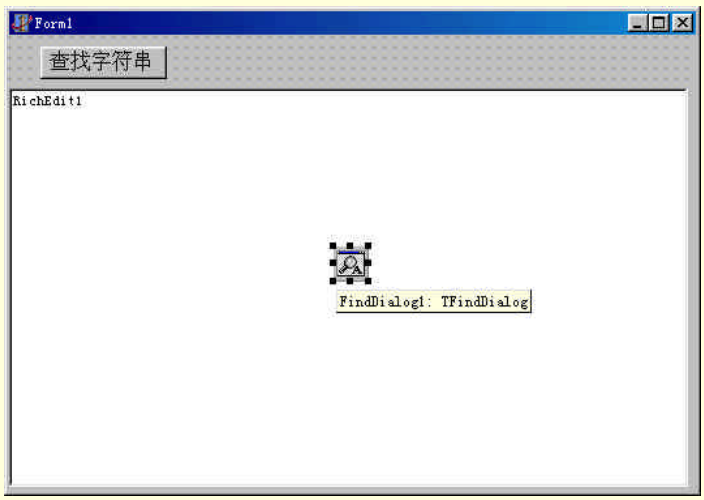


图 2-12 添加完控件后的窗体

Green Apple Data Center

2．添加代码

设计完上面的项目窗体，然后按照下面程序清单的提示添加完程序响应代码：

程序清单

```
unit FindString;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    RichEdit1: TRichEdit;
    FindDialog1: TFindDialog;
    procedure Button1Click(Sender: TObject);
    procedure FindDialog1Find(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
FindDialog1.Execute;
//显示对话框
end;

procedure TForm1.FindDialog1Find(Sender: TObject);
var
  FoundAt: LongInt;
  StartPos, ToEnd: integer;
  //变量声明
begin
  with RichEdit1 do
  begin
    if SelLength <> 0 then
      StartPos := SelStart + SelLength
    else
    StartPos := 0;
    ToEnd := Length(Text) - StartPos;
```

Green Apple Data Center

```
    FoundAt:=FindText(FindDialog1.FindText,StartPos,ToEnd,[stMatchCase]);
      if FoundAt <> -1 then
      begin
        SetFocus;
        SelStart := FoundAt;
        SelLength := Length(FindDialog1.FindText);
      end;
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.FindDialog1.FindText:='对话框';
//设置查找字符串
form1.FindDialog1.Options:=[frDown,frFindNext, frMatchCase, frWholeWord];
//设置对话框风格
end;
end.
```

程序说明：

在程序运行的初始阶段，首先执行窗体的初始化事件 FormCreate**，程序通过语句** form1.FindDialog1.FindText:='Wangxingjing';**和** form1.FindDialog1.Options:=[frDown, frFind Next, frMatchCase, frWholeWord];**设置了待查找的字符串和对话框的显示风格，如果用户单击了** Button1 **控件，那么就会调用** FindDialog **控件的** Execute **方法来显示一个对话框执行查找的操作，如果在用户指定的位置找到了指定的字符串，那么就会调用控件的** FindDialog1Find **方法选中找到的字符串。**

程序运行结果如图 2-13 **所示。**

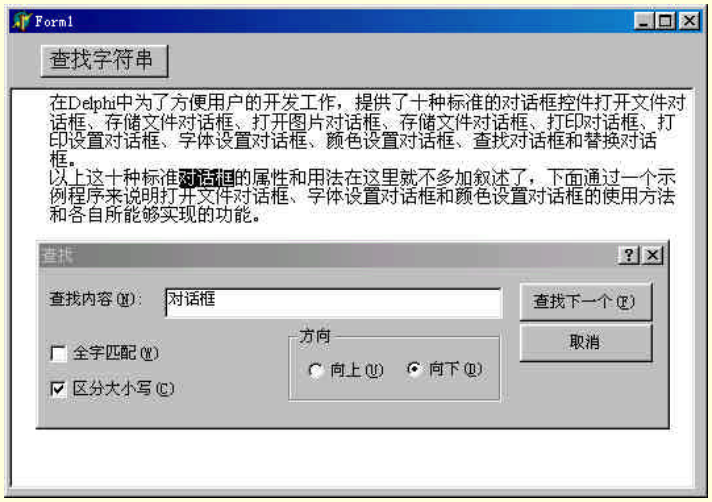


图 2-13　**程序运行结果**

其它的系统控件，如“打印”对话框、“打印设置”对话框、“替换”对话框、“打开图片”对话框和“存储图片”对话框，以及 Windows **帮助引擎等由于与“打开”等对话框的原理大致相同，在这里就不多加叙述，请读者自己在程序设计的过程加以体会。**

Green Apple Data Center

2.3 标准对话框的使用

在 Delphi 中为了方便用户的开发工作，提供了十种标准的对话框控件——打开文件对话框、存储文件对话框、打开图片对话框、存储文件对话框、打印对话框、打印设置对话框、字体设置对话框、颜色设置对话框、查找对话框和替换对话框。

以上这十种标准对话框的属性和用法在这里就不多加叙述了，下面通过一个示例程序来说明打开文件对话框、字体设置对话框和颜色设置对话框的使用方法和各自所能够实现的功能。

注意：

- 在下面的这个示例程序中，用户在程序的运行过程中，可以有选择的打开一个以*.txt 为文件后缀的文本文件，并且在程序的运行过程中，还可以改变文本的显示形式，如字体和颜色的设置等。

程序制作的具体步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项；

在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上的 Standard 选项和 Dialogs 选项中分别的选取 Button 按钮控件（四个）、Memo 控件、打开文件对话框控件、保存文件对话框控件、字体设置对话框控件和颜色设置对话框控件，并且把它们放置到空白的窗体上。添加控件后的窗体如图 2-14 所示。



图 2-14 添加控件后的窗体

其中窗体和控件的属性设置如下所示：

```
object Form1: TForm1
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
```

Green Apple Data Center

```
OnCreate = FormCreate
PixelsPerInch = 96
TextHeight = 13
object Memo1: TMemo
  Left = 8
  Top = 48
  Width = 361
  Height = 281
  Lines.Strings = ('Memo1')
  TabOrder = 0
end
object Button1: TButton
  Left = 8
  Top = 8
  Width = 75
  Height = 25
  Caption = '打开文件'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentFont = False
  TabOrder = 1
  OnClick = Button1Click
end
object Button2: TButton
  Caption = '保存文件'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentFont = False
  TabOrder = 2
  OnClick = Button2Click
end
object Button3: TButton
  Caption = '字体设置'
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  ParentFont = False
  TabOrder = 3
  OnClick = Button3Click
```

Green Apple Data Center

```
  end
  object Button4: TButton
    Caption = '颜色设置'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -13
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 4
    OnClick = Button4Click
  end
  object OpenDialog1: TOpenDialog
    Left = 80
    Top = 120
  end
  object SaveDialog1: TSaveDialog
    Left = 152
    Top = 176
  end
  object FontDialog1: TFontDialog
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -11
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    MinFontSize = 0
    MaxFontSize = 0
    Left = 216
    Top = 224
  end
  object ColorDialog1: TColorDialog
    Ctl3D = True
    Left = 264
    Top = 280
  end
end
```

2．程序的初始化

在上一步中添加的 ActiveX 控件的作用如下所示：

- 按钮“ Open ”：打开一个指定的文本文件；
- 按钮“ Save ”：保存本次编辑的文本文件；
- 按钮“ Font ”：设置文本控件中文本显示的字体；
- 按钮“ Color ”：设置文本控件中文本显示的背景颜色；
- Memo 控件：充当文本显示的容器；

而其余三个对话框控件的功能则是显示相应的对话框，在程序的设计过程中，要在窗体的 FormCreate()事件中添加对各个控件的初始化代码。

Green Apple Data Center

在程序的设计过程中，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个如图 2-15 所示的代码窗口。

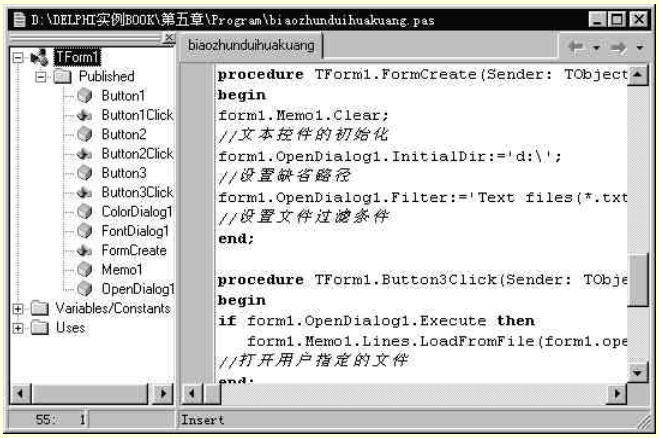


图 2-15　弹出代码窗口

把光标移动到窗体 FormCreate()事件的处理过程中，并且在其中添加如下所示的程序初始化代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Memo1.Clear;
//文本控件的初始化
form1.OpenDialog1.InitialDir:='d:\';
//设置打开文件的缺省路径
form1.OpenDialog1.Filter:='Text files(*.txt)|*.txt';
//设置打开文件过滤条件
form1.SaveDialog1.InitialDir:='d:\';
//设置保存文件的缺省路径
form1.SaveDialog1.Filter:='Text files(*.txt)|*.txt';
//设置保存文件的文件过滤条件
end;
```

程序说明：

在程序运行的初期，程序首先执行窗体 FormCreate()事件中的代码，即通过语句 form1.Memo1.Clear;来清空文本控件中的显示内容，然后通过语句 form1.OpenDialog1. InitialDir:= 'd:\';来设置打开文件的缺省路径为“ d:\ ”，最后设置文件的过滤条件。

3．响应按钮事件

为了能够实现各个按钮预计的功能，在程序的设计过程中，用鼠标的左键双击响应的按钮，在弹出的代码窗口中添加相应的代码即可。

为“ 打开文件 ”按钮添加代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
if form1.OpenDialog1.Execute then
    form1.Memo1.Lines.LoadFromFile(form1.opendialog1.filename);
//打开用户指定的文件
end;
```

程序说明：

在程序的运行过程中，当用户在‘ 打开文件 ’按钮上单击鼠标的左键时，就会自动的激活控件的 Button3Click()事件，如果以前没有任何已经打开的“ 打开文件 ”对话框，程序通过代码 form1.Memo1.Lines.LoadFromFile(form1.opendialog1.filename);来打开一个“ 打开文件 ”对话框对话框。

Green Apple Data Center

Windows 系统提供的典型的“ 打开文件 ”对话框如图 2-16 所示，它是经过封装的标准的对话框控件。

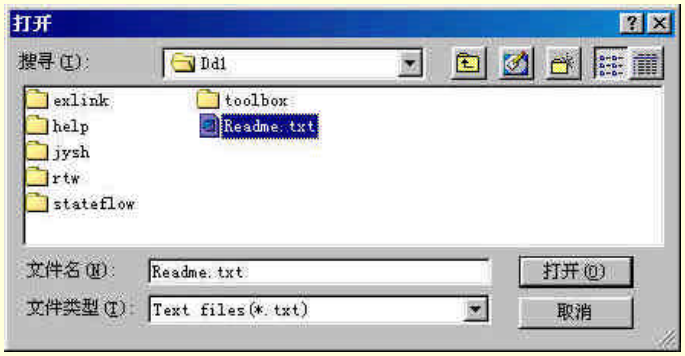


图 2-16　标准“ 打开文件 ”对话框

同理，为“ 保存文件 ”按钮添加代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
if form1.SaveDialog1.Execute then
    form1.Memo1.Lines.SaveToFile(form1.SaveDialog1.filename);
//保存用户指定的文件
end;
```

程序说明：

在程序的运行过程中，当用户在‘ 保存文件 ’按钮上单击鼠标的左键时，就会自动的激活控件的 Button4Click() 事件，如果以前没有任何已经打开的“ 保存文件 ”对话框，程序通过代码 form1.Memo1.Lines.SaveToFile(form1.savedialog1.filename);来打开一个“ 保存文件 ”对话框对话框。

Windows 系统提供的典型的“ 保存文件 ”对话框如图 2-17 所示，它也是经过封装的标准的对话框控件。

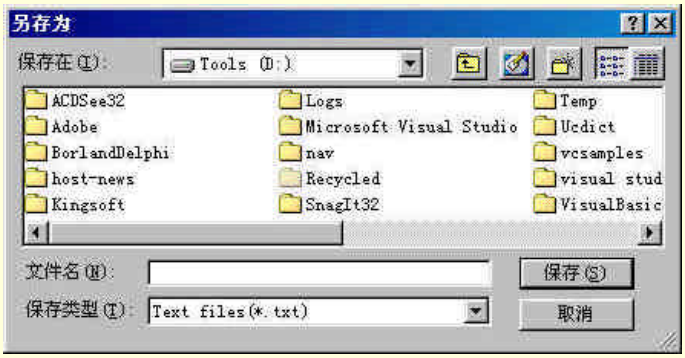


图 2-17　标准“ 保存文件 ”对话框

下面，以“ 字体设置 ”按钮为例来说明字体设置对话框控件，代码如下：

```
procedure TForm1.Button3Click(Sender: TObject);
begin
form1.FontDialog1.Font:=form1.Memo1.Font;
//初始化“ 字体 ”对话框的字体
if form1.FontDialog1.Execute then
    form1.Memo1.Font:=form1.FontDialog1.Font;
//设置文本框的字体
end;
```

程序说明：

在程序的运行过程中，当用户在‘ 打开文件 ’按钮上单击鼠标的左键时，就会自动的激活控件的 Button1Click()

Green Apple Data Center

事件，首先程序通过代码 form1.FontDialog1.Font:=form1. Memo1.Font;来初始化打开的字体设置对话框的字体属性，然后通过一个条件判断语句来设置当前文本框中文本显示的字体。

一个典型的字体设置对话框如图 2-18 所示。



图 2-18　字体设置对话框

最后，为“颜色设置”按钮添加执行代码如下：

```
procedure TForm1.Button4Click(Sender: TObject);
begin
form1.ColorDialog1.Color:=form1.Memo1.Color;
//初始化"颜色"对话框的颜色
if form1.ColorDialog1.Execute then
    form1.Memo1.Color:=form1.ColorDialog1.Color;
//设置文本框的颜色
end;
```

程序说明：

程序的运行过程中，当用户在“颜色”按钮上单击鼠标的左键时，就会自动的激活控件的 Button2Click() 事件，首先程序通过代码 form1.Memo1.Color:=form1.ColorDialog1.Color;来初始化打开的颜色设置对话框的颜色属性，然后通过一个条件判断语句来设置当前文本框中文本显示的颜色。

一个典型的颜色设置对话框如图 2-19 所示。



图 2-19　字体设置对话框

注意：

Green Apple Data Center

Windows 还提供了其他的标准对话框控件，请读者自己下去熟悉它们的使用。例如图 2-20 所示为典型的“ 打印 ”对话框。

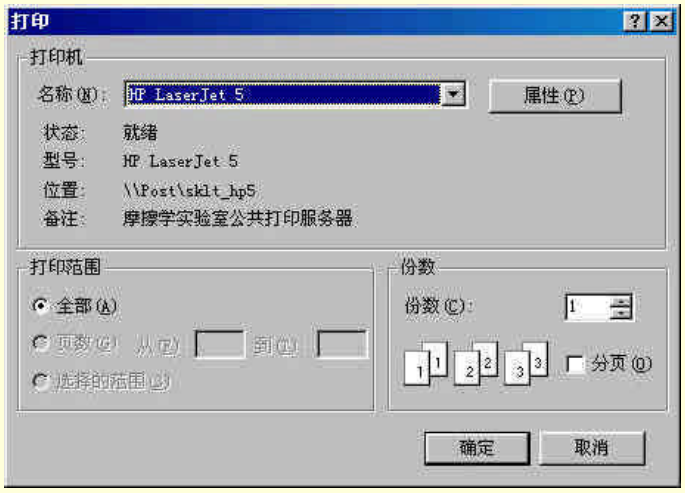


图 2-20　标准的“ 打印 ”对话框

4. 运行程序

参看附后的源程序代码，添加剩余的程序代码后，选择菜单“ File ”中的“ Save All ”选项，存储文件（这里读者可以保存项目为 P_Dialog，单元文件名为 Dialog）。

做完以上的工作后，按键盘上的功能键 F9 运行程序，在程序运行的初始画面中单击“ 打开文件 ”按钮，在屏幕上就会弹出一个“ 打开 ”的对话框，选择一个有效的文本文件后，单击“ 打开 ”按钮，返回到程序的运行画面，结果如图 2-21 所示。

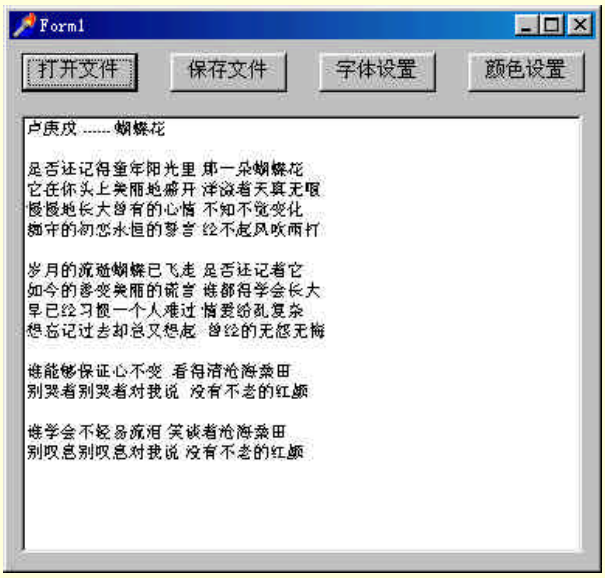


图 2-21　在程序中打开一个文本文件

在程序运行的过程中，打开一个文本文件后，就可以选择窗体上的“ 字体设置 ”和“ 颜色设置 ”按钮来改变文本框中文本的显示形式。

在本示例程序中，单击“ 字体设置 ”按钮，在弹出的“ 字体 ”对话框中选择“ 幼圆 ”10 号字体，单击“ 颜色设置 ”按钮，在弹出的“ 颜色 ”对话框中选择一个“ 淡兰色 ”的色彩，这样改变了文本显示形式的程序运行结果如图 2-22 所示。

Green Apple Data Center

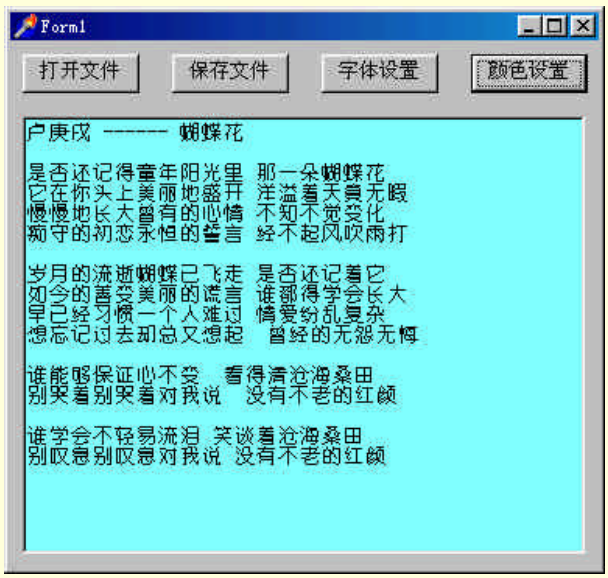


图 2-22 文本的不同显示形式

下面是程序的完整源代码：

程序清单

```
unit dialog;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    FontDialog1: TFontDialog;
    ColorDialog1: TColorDialog;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
```

Green Apple Data Center

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Memo1.Clear;
//文本控件的初始化
form1.OpenDialog1.InitialDir:='d:\';
//设置打开文件的缺省路径
form1.OpenDialog1.Filter:='Text files(*.txt)|*.txt';
//设置打开文件过滤条件
form1.SaveDialog1.InitialDir:='d:\';
//设置保存文件的缺省路径
form1.SaveDialog1.Filter:='Text files(*.txt)|*.txt';
//设置保存文件的文件过滤条件
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
if form1.OpenDialog1.Execute then
    form1.Memo1.Lines.LoadFromFile(form1.opendialog1.filename);
//打开用户指定的文件
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
if form1.SaveDialog1.Execute then
    form1.Memo1.Lines.SaveToFile(form1.SaveDialog1.filename);
//保存用户指定的文件
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
form1.FontDialog1.Font:=form1.Memo1.Font;
//初始化"字体"对话框的字体
if form1.FontDialog1.Execute then
    form1.Memo1.Font:=form1.FontDialog1.Font;
```

Green Apple Data Center

```
//设置文本框的字体
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
form1.ColorDialog1.Color:=form1.Memo1.Color;
//初始化"颜色"对话框的颜色
if form1.ColorDialog1.Execute then
    form1.Memo1.Color:=form1.ColorDialog1.Color;
//设置文本框的颜色
end;
end.
```

2.4 小 结

在本章中，系统的介绍了程序运行过程中，人机交互的一种常用的手段——Windows 对话框的程序设计方法，在读者的程序开发实践中，可以利用 Delphi 系统提供的对话框函数以及 Delphi 提供的标准对话框集合，来设计和编制自己的应用程序。

最后，读者当然也可以自己定义对话框，这一点希望在以后的开发实践中自己加以体会。

Green Apple Data Center

第三章　菜单程序的开发和设计

在 Windows 操作系统（无论是在 Windows 98/2000 还是在 Windows NT）中，大量的采用了菜单技术。一个典型的菜单就是 Windows 中 IE 浏览器中的系统菜单，如图 3-1 所示为“收藏”菜单项，用户在其中可以方便地使用各种选项进行操作。

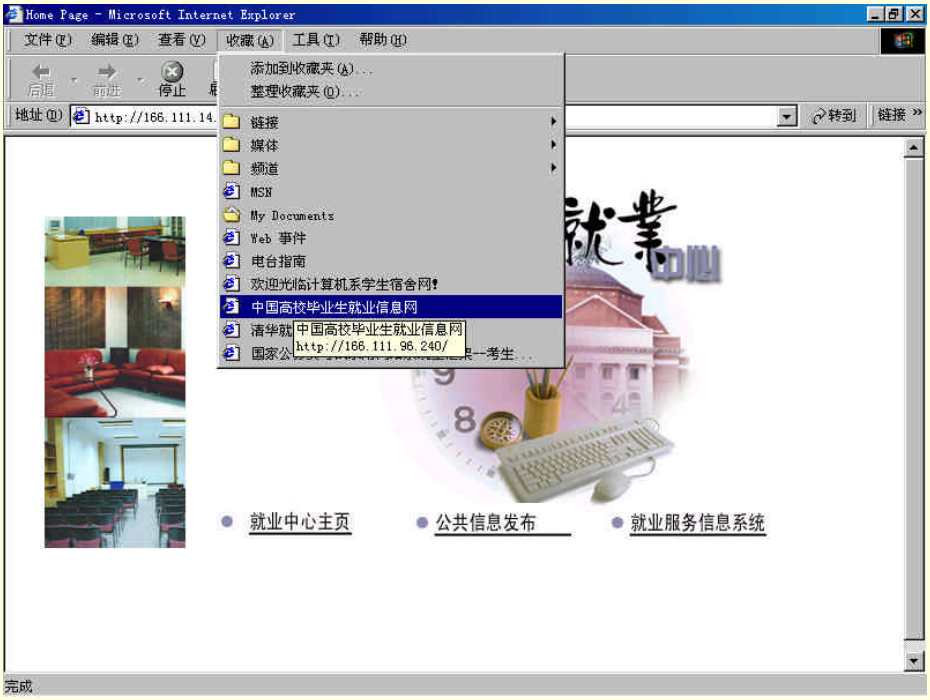


图 3-1　IE 浏览器中的系统菜单

在 Delphi 中有两种菜单设计器——MainMenu Designer（主菜单设计器）和 PopMenu Designer（弹出菜单设计器），它们都是不可见控件。

注意：

- 所谓的不可见，指的是在程序运行的过程中不可见，并不是程序设计过程中的不可见。

菜单也是可视化编程的特色之一，从某种意义上讲，也是衡量应用程序功能的一个方面，合理的利用菜单等 Windows 界面，可以大大的提高应用程序的人机交互功能，增加用户与程序之间的交流机会。

在本章中，我们仍然以几个比较常用的示例，将对菜单这种人机交互界面进行比较详细的讲解。下面就来分别的讲述一下如何利用主菜单设计器和弹出菜单设计器设计系统菜单和弹出式菜单。

3.1　主菜单基本概述

主菜单设计器位于 Delphi 控件工具栏中 Standard 选项下的最左侧，它的主要功能是设计系统菜单，或者叫做下拉式菜单。

3.1.1　创建菜单项

下面将会通过一个示例来说明如何在 Delphi 中创建一个菜单项，在这个菜单项中，有一个主菜单，而这个主菜单又由五个子菜单项组成，但是它们目前还不能够实现任何功能，具体的步骤如下：

Green Apple Data Center

1．添加控件

按照前面的方法新建立一个程序项目（Application Project），首先从控件工具栏上选择“ MainMenu（主菜单设计器）”，然后把它添加到空白的窗体上，在实际的操作过程中，可以通过两种方法向窗体上添加“ 主菜单设计器 ”：

- 在控件工具栏上单击鼠标的左键，然后把鼠标移动到空白的窗体上，按下的同时移动鼠标，在窗体上就会绘制出一个“ 主菜单设计器 ”；
- 在控件工具栏上用鼠标的左键双击“ 主菜单设计器 ”的图标，系统就会自动的在空白的窗体上放置一个“ 主菜单设计器 ”，但是这样做通常需要用户手动的调节控件的位置。

如图 3-2 所示即为放置了一个“ 主菜单设计器 ”的窗体。

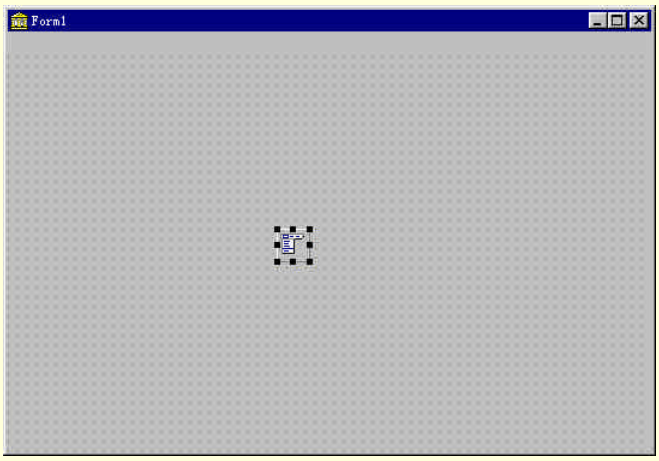


图 3-2　放置了“ 主菜单设计器 “ 的窗体

虽然在窗体上放置了“ 主菜单设计器 ”，但是只有激活了“ 主菜单设计器 ”才能够对主菜单进行设计等操作，激活“ 主菜单设计器 ”的方法也有两种：

- 用鼠标的左键在“ 主菜单设计器 ”上双击，在屏幕上就会弹出一个空白的主菜单设计窗口；
- 在控件的属性列表中用鼠标的左键单击 Items 属性右侧属性输入框中的按钮也可以打开主菜单设计器。

2．添加菜单项

在刚刚添加的“ 主菜单设计器 ”上双击鼠标的左键，就会弹出一个如图 3-3 所示的主菜单设计窗口。

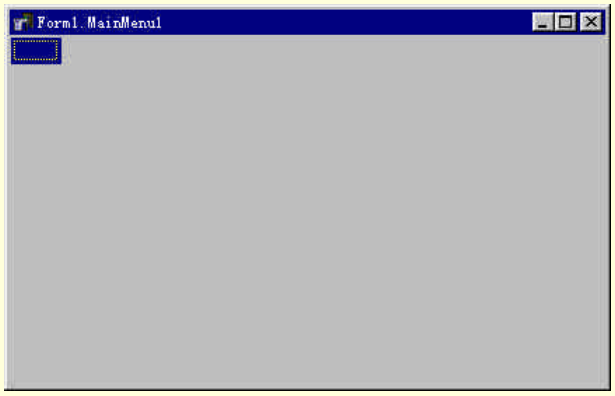


图 3-3　打开的主菜单设计器

在主菜单设计器中对主菜单及其子菜单项的属性设置如下所示，读者可以打开项目文件所对应的窗体文件（以*.dfm 为文件后缀）来观察程序中对窗体及各个控件的属性设置，由于“ 主菜单设计器 ”也属于控件类，所以相应菜单项的属性设置也可以在窗体文件中观察到。

```
object Form1: TForm1
  Left = 192
```

Green Apple Data Center

```
  Top = 107
  Width = 544
  Height = 375
  Caption = 'Form1'
  Menu = MainMenu1
  object MainMenu1: TMainMenu
    Left = 464
    Top = 264
    object File1: TMenuItem
      Caption = 'File'
      object New1: TMenuItem
        Caption = 'New'
      end
      object Open1: TMenuItem
        Caption = 'Open'
      end
      object Close1: TMenuItem
        Caption = 'Close'
      end
      object Save1: TMenuItem
        Caption = 'Save'
      end
      object Exit1: TMenuItem
        Caption = 'Exit'
      end
    end
  end
end
```

经过以上设置的菜单如图 3-4 **所示。**

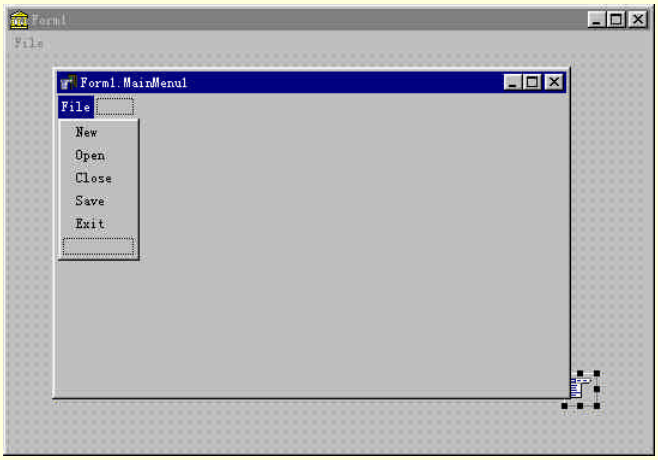


图 3-4　**设置属性值后的主菜单**

经过这样设置的“主菜单设计器”拥有一个主菜单项“File**”，在这个菜单项下有五个子菜单项——“**New、Open、Close、Save **和** Exit**”。**

3．**修改菜单项**

值得注意的是，以上的设计菜单的方法只是一种通用的方法，我们也可以通过在程序设计过程中修改项目

Green Apple Data Center

文件的窗体文件来达到设计菜单的目的，如上例所示，在主菜单“ File ”下有五个子菜单项，我们可以首先打开程序的窗体文件，然后在子菜单项 Save 和 Exit 之间添加如下内容：

```
object Save2: TMenuItem
Caption = 'Save as'
end
object About1: TMenuItem
Caption = 'About'
end
```

保存窗体文件，重新打开项目文件，在程序的设计阶段用鼠标的左键单击菜单项“ File ”，结果如图 3-5 所示。

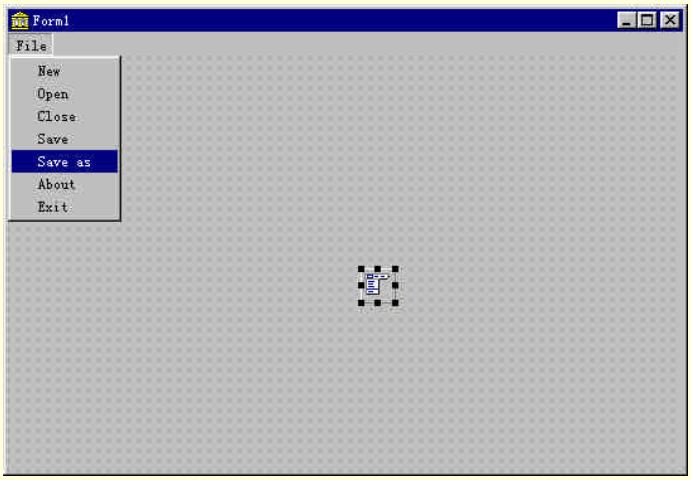


图 3-5　修改后的菜单项

3.1.2　设置分隔条

在菜单系统的设计中，经常可以看到所谓的将菜单项分组的技术，即在同一个菜单项下将一个几个子菜单项组合成一个集合，从而用分隔条相互的隔离开来，如图 3-6 所示即为一个典型的分隔条设置示例。

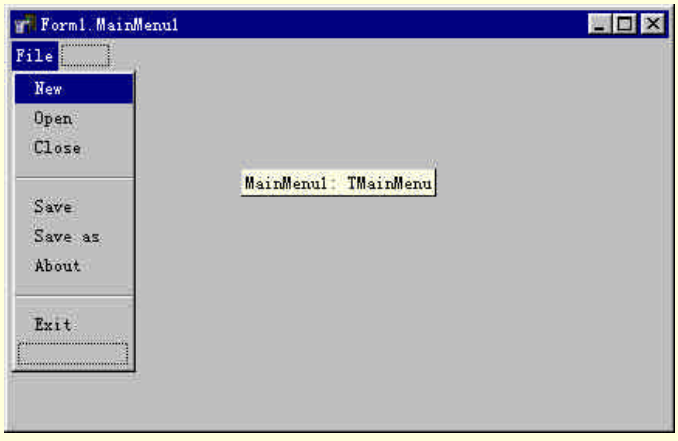


图 3-6　菜单项中的分隔条

其实分隔条的设计也是相当简单的，只要在菜单项的 Caption 属性中设置为“ - ”即可，如图 3-6 所示的菜单项分隔条属性设置如下：

```
object New1: TMenuItem
Caption = 'New'
end
```

Green Apple Data Center

```
object Open1: TMenuItem
Caption = 'Open'
end
object Close1: TMenuItem
Caption = 'Close'
end
object N1: TMenuItem
Caption = '-'
end
        //设置分割条
object Save1: TMenuItem
Caption = 'Save'
end
object Save2: TMenuItem
Caption = 'Save as'
end
object About1: TMenuItem
Caption = 'About'
end
object N2: TMenuItem
Caption = '-'
end
      //设置分割条
object Exit1: TMenuItem
Caption = 'Exit'
end
```

3.1.3 设置快捷键

一般来说，常用的菜单项都有自己的快捷键（加速键或者热键），为菜单项设置一个合适的快捷键，可以提高用户访问命令和使用菜单项的效率，如图 3-7 所示为 Word 中“编辑”菜单，基本上每一个菜单项都有自己的快捷键方式。



图 3-7 有快捷键的菜单项

在 Delphi 中可以为每一个子菜单项设置快捷键：

在 Delphi 中，通过设置子菜单项的 Caption 属性可以为子菜单项设置加速键，在设置 Caption 属性时，在需要设置为加速键的字母前面加一个“&”符号，这样菜单项中的该字母就会自动的加上一个下划线，在程序

Green Apple Data Center

运行过程中用户按下 Alt 键和该字母的组合键就可以实现选中子菜单项的功能。

如图 3-8 所示即为设置了加速键的子菜单项——New，在程序运行的过程中，用户随时的按下 Alt+N 键就可以访问子菜单项 New。

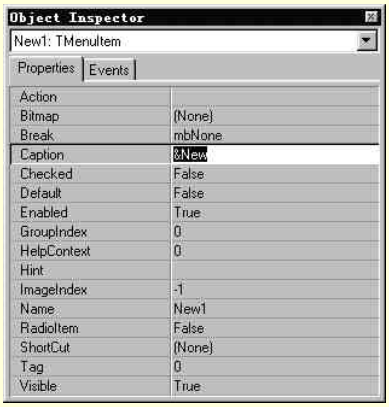


图 3-8　设置加速键

当为子菜单项设置热键时，操作方法与以上略有不同。

首先用鼠标选中想要设置热键的子菜单项，然后用鼠标的左键单击这个菜单项，返回到子菜单项的属性列表中，单击属性列表中 ShortCut 属性输入框右侧的下拉按钮，在弹出的热键列表中选择一个合适的热键即可。

如图 3-9 所示为设置了加速键和热键的菜单效果。

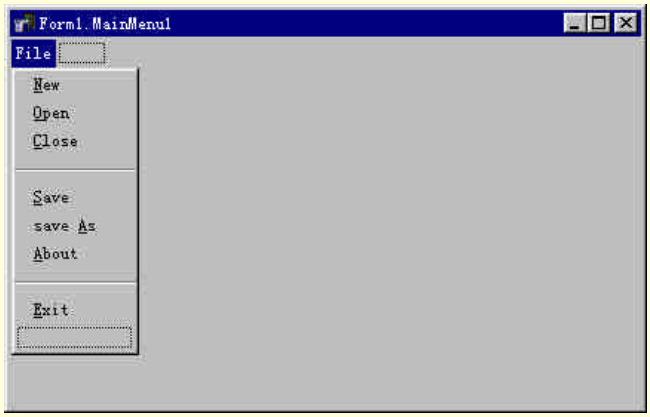


图 3-9　设置了加速键和热键的菜单项

其中各个菜单项的属性设置如下所示：

```
object File1: TMenuItem
Caption = 'File'
object New1: TMenuItem
Caption = '&New'
end
object Open1: TMenuItem
Caption = '&Open'
end
object Close1: TMenuItem
Caption = '&Close'
end
object N1: TMenuItem
Caption = '-'
end
```

Green Apple Data Center

```
object Save1: TMenuItem
Caption = '&Save'
end
object Save2: TMenuItem
Caption = 'save &As'
end
object About1: TMenuItem
Caption = '&About'
end
object N2: TMenuItem
Caption = '-'
end
object Exit1: TMenuItem
Caption = '&Exit'
end
end
```

3.1.4 设置复选标志

所谓的复选标志，指的是当用户选择菜单项中的某一个子菜单项时，在相应的菜单项的前面就会显示一个选中的标记符号“√”，而当用户再次单击这个菜单项时，复选标志又会消失。

复选标志可以在程序的设计阶段通过子菜单项的属性来加以设置，方法是在程序的设计阶段首先用鼠标选中子菜单项，然后把鼠标移动到菜单项对应的属性列表框中，把 Checked 属性设置为 True，，如图 3-10 所示。那么在程序的运行过程中，该菜单项的前面就会显示一个复选标志。



图 3-10 设置菜单的复选标志

当然，我们也可以在程序的运行过程中通过代码来动态的设置菜单项的复选标志，首先在窗体的 FormCreate()事件中加入对菜单项的初始化代码如下所示：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Open1.Checked:=true;
//初始化菜单项的状态
end;
```

以上的代码在程序运行的初期就把子菜单项 New 设置为复选状态，然后在程序的设计阶段用鼠标的左键单击子菜单项 New，在弹出的代码窗口中添加如下所示的相应菜单事件的代码：

```
procedure TForm1.Open1Click(Sender: TObject);
begin
```

Green Apple Data Center

```
form1.Open1.Checked:=not form1.Open1.Checked;
//鼠标单击后改变菜单项的状态
end;
```

这段代码能够实现子菜单项复选状态之间的互换，即用鼠标单击子菜单项会改变菜单项的选中状态。以上代码的运行结果如图 3-11 所示。

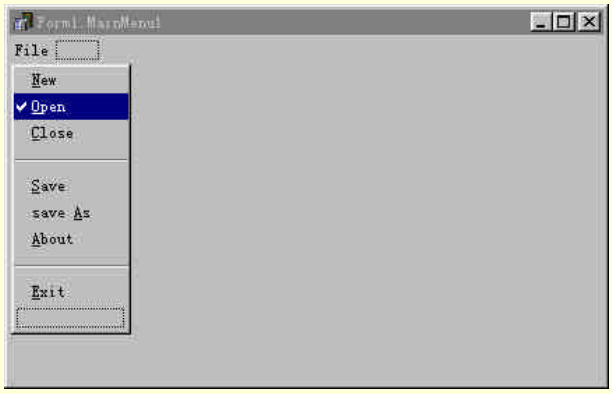


图 3-11　动态改变菜单的复选状态

3.1.5　菜单项有效状态设置

在其它的应用程序中，经常的看到在一些菜单中，程序可以动态的设置菜单的有效状态，即在一定的条件下，某个菜单处于无效的状态，在另外的条件下，菜单项又会变为有效的状态，一个十分明显的例子就是编辑菜单中 Paste 子菜单项，当剪贴板中没有数据时，Paste 就会处于无效的状态，一旦剪贴板上有数据，那么 Paste 就会立刻的变为有效的状态，在 Delphi 中也可以实现这种效果。

下面就讲述一下如何通过代码来在程序的运行过程中动态的改变菜单项的有效状态。

首先按照如图 3-12，按照前面所讲的方法设计一个比较完整的系统下拉式菜单，包括另外两个主菜单 Edit 和 Help。

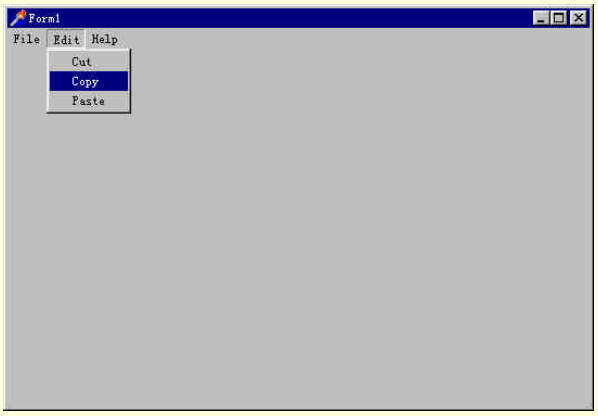


图 3-12　设计完成的下拉式菜单

然后，在程序的设计过程中，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口中的 FormCreate()事件处理过程中，并且添加如下所示的程序初始化代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Copy1.Enabled:=true;
form1.Paste1.Enabled:=false;
//初始化菜单项的有效状态
end;
```

在程序运行的初期，在剪贴板上还没有可以利用的数据，所以要通过语句 form1.Paste1.Enabled:=false;把子

Green Apple Data Center

菜单项 Paste 设置为无效的状态。

在程序运行的过程中，当用户用鼠标的左键单击子菜单项 Copy 时，系统的剪贴板就会带有数据，所以子菜单项 Paste 就要变为有效的状态。

在代码窗口中找到子菜单项的响应事件 Copy1Click()，并且添加如下的菜单响应代码：

```
procedure TForm1.Copy1Click(Sender: TObject);
begin
form1.Copy1.Enabled:=false;
form1.Paste1.Enabled:=true;
//鼠标单击后改变菜单项的状态
end;
```

做完以上的工作后，存储文件（保存项目文件为 App3_1，单元文件为 mainmenu），按键盘上的功能键 F9 运行程序，结果如图 3-13 所示。

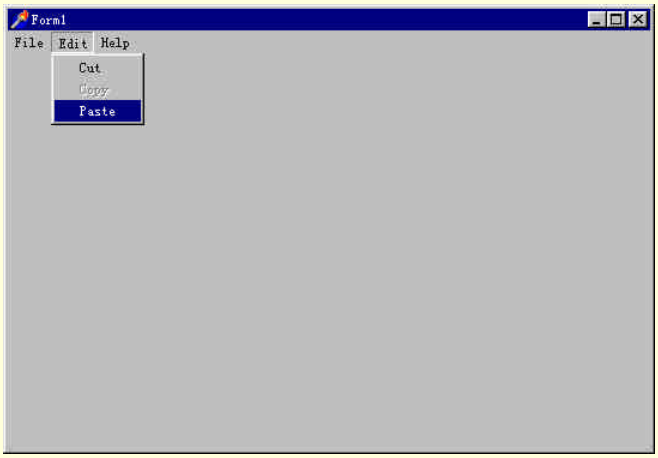


图 3-13 主菜单应用程序界面

3.2 主菜单的应用

本节，我们将向读者介绍一个主菜单的设计实例，实际是一个“ 简易记事本 ”，在其中可以实现以下的功能：

- 能够新建一个文本文件，并且能够把用户的修改结果保存到文本文件中；
- 能够打开一个已经存在的文本文件，而且在程序运行的过程中能够加以修改；
- 在程序运行的过程中，应用程序能够同剪贴板之间进行简单的数据交换等。

制作能够实现以上功能的文本处理应用程序的具体步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项；

在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Standart ”选项后，在“ MainMenu ”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个主菜单设计器。接着向窗体上添加一个 OpenDialog 控件、一个 SaveDialog 控件和一个 Memo 控件。

各个控件的功能将在后面分别的加以介绍，添加控件后的窗体如图 3-14 所示。

Green Apple Data Center



图 3-14 添加控件后的窗体

其中添加到窗体上的各个控件的功能如下所示：

- MainMenu 控件的作用是为应用程序设计一个下拉式主菜单；
- 而 OpenDialog 控件的作用是显示一个“打开”对话框，用户在其中可以选择一个有效的文本文件；
- SaveDialog 控件的作用是为用户存储文件提供存储的路径和文件名；
- Memo 控件的作用是为用户在程序运行过程中进行文本编辑提供一个容器。

在下拉式主菜单中只包括三个主菜单项“File”、“Edit”和“Style”，在各个主菜单项的下面又包括有几个子菜单项，菜单的结构如表 3-1 所示。

表 3-1 菜单项的结构表

主菜单项	子菜单项
文件	新文件
	打开文件
	保存
	退出
编辑	减切
	拷贝
	粘贴
	删除
	全选
格式	左对齐
	中对齐
	右对齐
	黑体
	斜体
	下划线

有关菜单设计的属性设置如下：

```
object File1: TMenuItem
  Caption = '文件'
  object New1: TMenuItem
    Caption = '新文件'
  end
  object Open1: TMenuItem
    Caption = '打开文件'
```

Green Apple Data Center

```
      Checked = True
      OnClick = Open1Click
    end
    object N1: TMenuItem
      Caption = '-'
    end
    object Save1: TMenuItem
      Caption = '保存'
    end
    object N2: TMenuItem
      Caption = '-'
    end
    object Exit1: TMenuItem
      Caption = '退出'
    end
  end
  object Edit1: TMenuItem
    Caption = '编辑'
    object Cut1: TMenuItem
      Caption = '减切'
    end
    object Copy1: TMenuItem
      Caption = '拷贝'
    end
    object Paste1: TMenuItem
      Caption = '粘贴'
    end
    object Delete1: TMenuItem
      Caption = '删除'
    end
    object Selectall1: TMenuItem
      Caption = '全选'
    end
  end
  object Style1: TMenuItem
    Caption = '格式'
    object Bold1: TMenuItem
      Caption = '黑体'
    end
    object Italic1: TMenuItem
      Caption = '斜体'
    end
    object Underline1: TMenuItem
      Caption = '下划线'
    end
    object N3: TMenuItem
```

Green Apple Data Center

```
      Caption = '-'
    end
    object Left1: TMenuItem
      Caption = '左对齐'
    end
    object Center1: TMenuItem
      Caption = '中对齐'
    end
    object Right1: TMenuItem
      Caption = '右对齐'
    end
  end
```

经过以上的属性设置后的下拉式主菜单如图 3-15 所示。



图 3-15　设计完成的下拉式主菜单

2．程序的初始化

程序的初始化过程，实际上就是对窗体 FormCreate()事件的初始化，在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Memo1.Clear;
//清空文本框
form1.OpenDialog1.InitialDir:='c:\windows';
form1.SaveDialog1.InitialDir:='c:\wondows';
//设置对话框的缺省路径
form1.OpenDialog1.Filter:='Text Files(*.txt)|*.txt';
form1.SaveDialog1.Filter:='Text Files(*.txt)|*.txt';
//设置文件过滤器
end;
```

在程序运行的初期，首先执行窗体 FormCreate()事件中的代码，即通过语句 form1.Memo1.Clear;来清空文本框中的显示内容，然后设置 OpenDialog 控件和 SaveDialog 控件的缺省路径为 c:\windows，最后通过两条语句

```
form1.OpenDialog1.Filter:='Text Files(*.txt)|*.txt';
form1.SaveDialog1.Filter:='Text Files(*.txt)|*.txt';
```

来设置文件的过滤条件为'Text Files(*.txt)|*.txt';，即在“打开”和“另存为…”对话框中只能够显示以*.txt 结尾的文本文件。

经过代码初始化后的窗体及其控件如图 3-16 所示。

Green Apple Data Center



图 3-16 初始化后的窗体及其控件

3. 响应“文件”菜单

在“文件”菜单下有四个子菜单项——“新文件”、“打开文件”、“保存”和“退出”，它们的功能如下所示：

- “新文件”菜单项：新建一个文本文件，同时清空文本框；
- “打开文件”菜单项：打开一个已经存在的文本文件；
- “保存”菜单项：把当前文本框中的内容存储到一个文本文件中去；
- “退出” 菜单项：结束程序的运行。

为了能够实现子菜单项“打开文件”项的功能，在程序的设计过程中用鼠标的左键单击菜单“文件”/“打开文件”，在屏幕上就会弹出一个代码窗口，在其中可以添加对子菜单项“打开文件”项的响应代码如下所示：

```
procedure TForm1.Open1Click(Sender: TObject);
begin
form1.OpenDialog1.Execute;
//打开一个对话框
form1.Memo1.Lines.LoadFromFile(form1.OpenDialog1.FileName);
//打开文本文件
end;
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击菜单“File”/“Open”时，就会激活菜单的 Open1Click() 事件，然后程序通过语句 form1.OpenDialog1.Execute;来显示一个“打开”对话框，用户可以在其中选择一个有效的文本文件，单击“打开”按钮后，在文本框中就会显示出打开的文本文件的内容，具体的打开文件的操作是通过 Memo 控件来完成的，实现的语句为 form1.Memo1.Lines.LoadFromFile(form1.OpenDialog1.FileName);。

我们在前面已经讲解过类似菜单 Open 项的代码添加过程，不再赘述，其余菜单项的代码请参见附后的源程序代码。

4. 响应“编辑”菜单

在“编辑”菜单下有五个子菜单项——“剪切”、“拷贝”、“粘贴”、“删除”和“全选”，它们的功能如下所示：

- 菜单项“剪切”：把当前文本框中选中的数据剪贴到剪贴板上；
- 菜单项“拷贝”：把当前文本框中选中的数据复制到剪贴板上；
- 菜单项“粘贴”：把当前剪贴板上的数据粘贴到文本框的当前位置上；
- 菜单项“删除”：把当前文本框中的选中数据清除；
- 菜单项“全选”：选中当前剪贴板上的所有数据

下面仅以子菜单“拷贝”项来说明“编辑”菜单项的代码添加过程，其余子菜单的代码添加过程请参见附后的源程序代码。

Green Apple Data Center

为了能够实现子菜单项“拷贝”项的功能，在程序的设计过程中用鼠标的左键单击菜单“编辑”/“拷贝”，在屏幕上就会弹出一个代码窗口，在其中可以添加对子菜单项“拷贝”项的响应代码如下所示：

```
procedure TForm1.Copy1Click(Sender: TObject);
begin
form1.Memo1.CopyToClipboard;
//将选中的数据复制到剪贴板上
end;
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击菜单“Edit”/“Copy”时，就会激活菜单的 Copy1Click () 事件，然后程序通过语句

form1.Memo1.CopyToClipboard;

把当前文本框中的选中数据复制到剪贴板上。

5．响应“格式”菜单

在“格式”菜单下有六个子菜单项——“左对齐”、“中对齐”、“右对齐”、“黑体”、“斜体”和“下划线”，它们的功能如下所示：

- 子菜单“左对齐”：将当前文本框中的文本以左对齐方式显示；
- 子菜单“中对齐”：将当前文本框中的文本以居中对齐方式显示；
- 子菜单“右对齐”：将当前文本框中的文本以右对齐方式显示；
- 子菜单“黑体”：当前文本框中的文本以黑体显示；
- 子菜单“斜体”：当前文本框中的文本以斜体显示；
- 子菜单“下划线”：当前文本框中的文本以下划线形式显示。

下面仅以子菜单“中对齐”项和“黑体”来说明“格式”菜单项的代码添加过程，其余子菜单的代码添加过程请参见附后的源程序代码。

为了能够实现子菜单项“中对齐”项的功能，在程序的设计过程中用鼠标的左键单击菜单“格式”/“中对齐”，在屏幕上就会弹出一个代码窗口，在其中可以添加对子菜单项“中对齐”项的响应代码如下所示：

```
procedure TForm1.Center1Click(Sender: TObject);
begin
form1.Memo1.Alignment:=taCenter;
//设置文本居中对齐
end;
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击菜单“格式”/“中对齐”时，就会激活菜单的 Center1Click ()事件，然后程序通过语句 form1.Memo1.Alignment:=taCenter;来设置文本框中的文本为居中对齐方式。

为了能够实现子菜单项“黑体”项的功能，在程序的设计过程中用鼠标的左键单击菜单“格式”/“黑体”，在屏幕上就会弹出一个代码窗口，在其中可以添加对子菜单项“黑体”项的响应代码如下所示：

```
procedure TForm1.Bold1Click(Sender: TObject);
begin
form1.Bold1.Checked:=not form1.Bold1.Checked;
//设置复选状态
if form1.Bold1.Checked then
    form1.Memo1.Font.Style:=form1.Memo1.Font.Style+ [fsBold]
    else
    form1.Memo1.Font.Style:=form1.Memo1.Font.Style- [fsBold];
    //设置字体是否以黑体显示
end;
```

程序说明

Green Apple Data Center

在程序运行的过程中，当用户用鼠标的左键单击菜单“ Style ”/“ Bold ”时，就会激活菜单的 Bold 1Click () 事件，首先通过语句

form1.Bold1.Checked:=not form1.Bold1.Checked;

来设置子菜单项“ 黑体 ”的复选状态，这样当用鼠标的左键单击子菜单“ 黑体 ”时，就会转换子菜单项的复选状态。

然后程序通过语句 form1.Bold1.Checked 来判断子菜单“ 格式 ”的复选状态，如果处于复选状态，那么通过语句 form1.Memo1.Font.Style:=form1.Memo1.Font.Style+ [fsBold]使得文本框中的文本以黑体显示，否则将通过语句

form1.Memo1.Font.Style:=form1.Memo1.Font.Style- [fsBold];

在文本框中正常的显示文本。

6. 运行程序

按照附后的源程序，添加剩余的程序代码后，选择菜单“ File ”中的“ Save All ”选项，在弹出的对话框中选择合适的文件名保存文件（这里我们保存项目文件为 App4_1，单元文件为 notepad）；

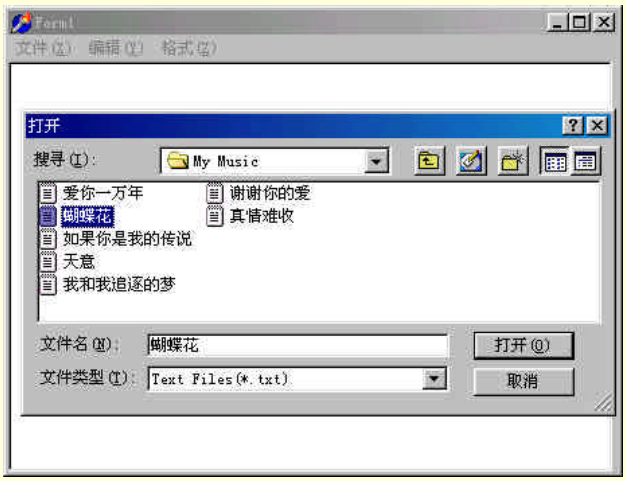


图 3-17 程序运行中的“ Style ”菜单

然后按键盘上的功能键 F9 运行程序，在程序运行的初始画面中，首先选择菜单“ 文件 ”/“ 打开文件 ”，在弹出的“ 打开文件 ”对话框（如图 3-17 所示）中选择一个有效的文本文件后，单击“ 打开 ”按钮，在程序运行的窗体中就显示了这个文本文件的内容，如图 3-18 所示；

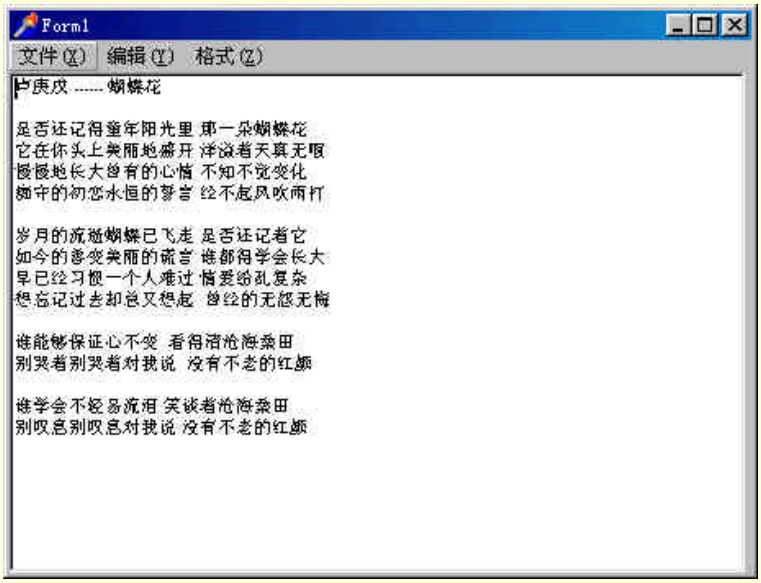


图 3-18 打开的一个文本

然后，选择“ 编辑 ”/“ 全选 ”选项，然后选择“ 格式 ”/“ 斜体 ”和“ 下划线 ”，文本变为运行的结果如图

Green Apple Data Center

3-19 所示。

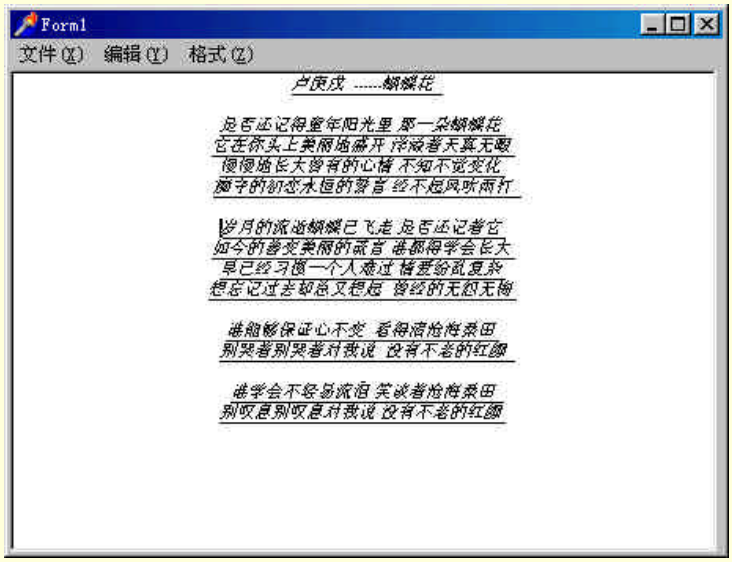


图 3-19 文本编辑效果

附程序完整的源代码如下所示：

程序清单

```
unit notepad;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Menus, ComCtrls, ImgList, ToolWin, StdCtrls;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    New1: TMenuItem;
    Open1: TMenuItem;
    Save1: TMenuItem;
    Exit1: TMenuItem;
    N1: TMenuItem;
    N2: TMenuItem;
    Style1: TMenuItem;
    Bold1: TMenuItem;
    Italic1: TMenuItem;
    Underline1: TMenuItem;
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    Edit1: TMenuItem;
    Cut1: TMenuItem;
```

Green Apple Data Center

```
    Copy1: TMenuItem;
    Paste1: TMenuItem;
    Delete1: TMenuItem;
    Selectall1: TMenuItem;
    N3: TMenuItem;
    Left1: TMenuItem;
    Center1: TMenuItem;
    Right1: TMenuItem;
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
    procedure Open1Click(Sender: TObject);
    procedure Copy1Click(Sender: TObject);
    procedure Center1Click(Sender: TObject);
    procedure Bold1Click(Sender: TObject);
    procedure Left1Click(Sender: TObject);
    procedure Right1Click(Sender: TObject);
    procedure Italic1Click(Sender: TObject);
    procedure Underline1Click(Sender: TObject);
    procedure Cut1Click(Sender: TObject);
    procedure Paste1Click(Sender: TObject);
    procedure Delete1Click(Sender: TObject);
    procedure Selectall1Click(Sender: TObject);
    procedure Save1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure New1Click(Sender: TObject);
    private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Memo1.Clear;
//清空文本框
form1.OpenDialog1.InitialDir:='c:\windows';
form1.SaveDialog1.InitialDir:='c:\windows';
//设置对话框的缺省路径
form1.OpenDialog1.Filter:='Text Files(*.txt)|*.txt';
form1.SaveDialog1.Filter:='Text Files(*.txt)|*.txt';
```

Green Apple Data Center

```
//设置文件过滤器
end;

procedure TForm1.Open1Click(Sender: TObject);
begin
form1.OpenDialog1.Execute;
//打开一个对话框
form1.Memo1.Lines.LoadFromFile(form1.OpenDialog1.FileName);
//打开文本文件
end;

procedure TForm1.Copy1Click(Sender: TObject);
begin
form1.Memo1.CopyToClipboard;
//将选中的数据复制到剪贴板上
end;

procedure TForm1.Center1Click(Sender: TObject);
begin
form1.Memo1.Alignment:=taCenter;
//设置文本居中对齐
end;

procedure TForm1.Bold1Click(Sender: TObject);
begin
form1.Bold1.Checked:=not form1.Bold1.Checked;
//设置复选状态
if form1.Bold1.Checked then
    form1.Memo1.Font.Style:=form1.Memo1.Font.Style+ [fsBold]
    else
    form1.Memo1.Font.Style:=form1.Memo1.Font.Style- [fsBold];
    //设置字体是否以黑体显示
end;

procedure TForm1.Left1Click(Sender: TObject);
begin
form1.Memo1.Alignment:=taLeftJustify;
//设置文本左对齐方式
end;

procedure TForm1.Right1Click(Sender: TObject);
begin
form1.Memo1.Alignment:=taRightJustify;
//设置文本右对齐方式
end;
```

Green Apple Data Center

```
procedure TForm1.Italic1Click(Sender: TObject);
begin
form1.Italic1.Checked:=not form1.Italic1.Checked;
//设置复选状态
if form1.Italic1.Checked then
    form1.Memo1.Font.Style:=form1.Memo1.Font.Style+ [fsItalic]
    else
    form1.Memo1.Font.Style:=form1.Memo1.Font.Style- [fsItalic];
    //设置字体是否以斜体显示
end;

procedure TForm1.Underline1Click(Sender: TObject);
begin
form1.UnderLine1.Checked:=not form1.UnderLine1.Checked;
//设置复选状态
if form1.UnderLine1.Checked then
    form1.Memo1.Font.Style:=form1.Memo1.Font.Style+ [fsUnderLine]
    else
    form1.Memo1.Font.Style:=form1.Memo1.Font.Style- [fsUnderLine];
    //设置字体是否以下划线形式显示
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
form1.Memo1.CutToClipboard;
//将选中的数据剪切到剪贴板上
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
form1.Memo1.PasteFromClipboard
//将当前剪贴板上的数据粘贴到当前位置上
end;

procedure TForm1.Delete1Click(Sender: TObject);
begin
form1.Memo1.ClearSelection;
//清除当前选中的数据
end;

procedure TForm1.Selectall1Click(Sender: TObject);
begin
form1.Memo1.SelectAll;
//选中控件上的全部数据
end;
```

Green Apple Data Center

```
procedure TForm1.Save1Click(Sender: TObject);
begin
form1.SaveDialog1.Execute;
//打开一个对话框
form1.Memo1.Lines.SaveToFile(form1.SaveDialog1.filename);
//存储到文本文件中
end;

procedure TForm1.Exit1Click(Sender: TObject);
begin
close;
//结束程序的运行
end;
procedure TForm1.New1Click(Sender: TObject);
begin
form1.Memo1.Clear;
//清空文本框
end;
end.
```

3.3 主菜单和弹出菜单综合应用

在通常情况下，一个大的应用程序都有一个主菜单系统，但是在程序的内部为了方便用户的使用，通常还需要给出弹出菜单。与主菜单的激活方式不同，弹出菜单的激活方式为单击鼠标右键，一个典型的弹出式菜单如图 3-20 所示。



图 3-20 弹出式菜单

与下拉式主菜单的设计类似，“ 弹出式菜单设计器 ” 的位置也位于 Delphi 控件工具栏 Standart 选项。，通过下面所示的两种方法都可以向窗体上添加 “ 弹出式菜单设计器 ”，它们的效果是相同的：

- 在控件工具栏上单击鼠标的左键，然后把鼠标移动到空白的窗体上，按下的同时移动鼠标，在窗体上就会绘制出一个 “ 弹出式菜单设计器 ”；
- 在控件工具栏上用鼠标的左键双击 “ 弹出式菜单设计器 ” 的图标，系统就会自动的在空白的窗体上放置一个 “ 弹出式菜单设计器 ”，但是这样做通常用户手动的调节控件的位置。

如图 3-21 所示即为放置了一个 “ 弹出式菜单设计器 “ 的窗体。

Green Apple Data Center

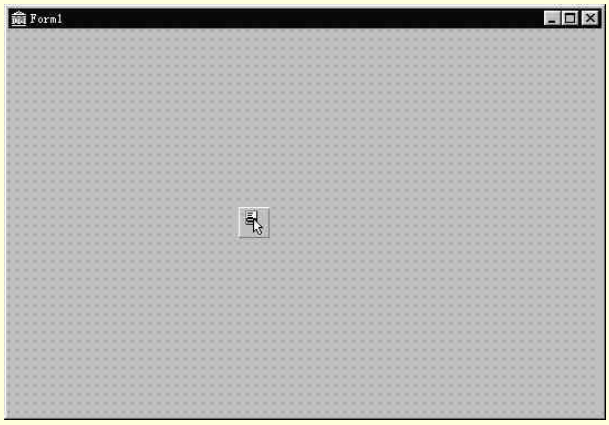


图 3-21　放置了“弹出式菜单 PopMenu”的窗体

虽然在窗体上放置了“弹出式菜单设计器”，但是只有激活了“弹出式菜单设计器”才能够对弹出式菜单进行设计等操作，激活“弹出式菜单设计器”的方法也有两种：

- 用鼠标的左键在“弹出式菜单设计器”上双击，在屏幕上就会弹出一个空白的弹出式菜单设计窗口；
- 在控件的属性列表中用鼠标的左键单击 Items 属性右侧属性输入框中的按钮也可以打开弹出式菜单设计器。

由于弹出式菜单的设计过程与下拉式菜单的设计过程基本相同，所以在这里对具体的设计过程就不多加赘述了。下面就以一个示例程序来说明弹出式菜单的完部设计和操作过程。

3.3.1　项目和设置

首先启动一个新的项目，选择菜单“File”中的“New Application”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“Standard”选项后，在“弹出式菜单设计器”的图标上双击鼠标的左键，这时空白的窗体上就会出现一个弹出式菜单设计器。

用鼠标的左键在“弹出式菜单设计器”上双击，在屏幕上就会弹出一个空白的弹出式菜单设计窗口，在其中如下设置弹出式菜单：

```
object PopupMenu1: TPopupMenu
Left = 184
Top = 24
object Cut1: TMenuItem
Caption = 'Cut'
end
object Copy1: TMenuItem
Caption = 'Copy'
end
object Paste1: TMenuItem
Caption = 'Paste'
end
object Delete1: TMenuItem
Caption = 'Delete'
end
object SelectAll1: TMenuItem
Caption = 'Select All'
end
end
```

设计完成的弹出式菜单如图 3-22 所示。

Green Apple Data Center

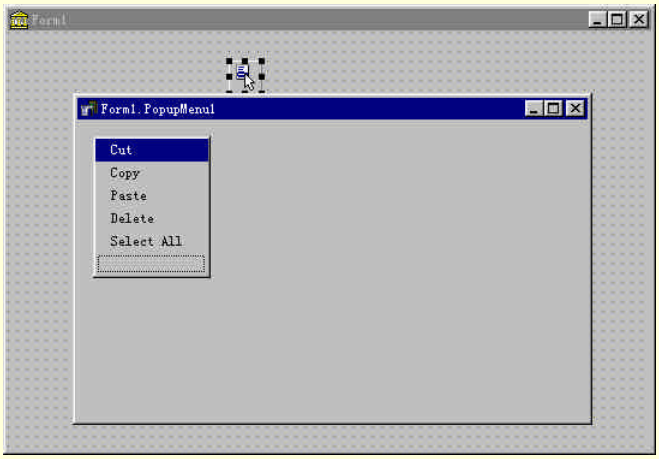


图 3-22　设计完成的弹出式菜单

经过以上属性设置后的弹出式菜单，在程序运行的过程中，单击鼠标的右键就会激活它，并且在其中包含有四个子菜单项——Cut、Copy、Paste、Delete 和 Select All。

3.3.2　弹出菜单的设计

1．结合主菜单应用

为了能够实现文本处理功能，需要向窗体上添加一个 Memo 控件。

然后，为了将弹出菜单和下拉菜单结合使用，我们再向窗体上添加一个主菜单 Style1（MainMenu）。其中主菜单上包括四个菜单项：全拼输入法（N1）、微软拼音输入法（N2）、郑码输入法（N3）和英文输入法（N4），设计如图 3-23 所示。

各个控件所能够实现的功能如下所示：

- Memo 控件：为文本处理提供一个容器；
- N1 控件：设置文本框中输入文本的方式为“全拼输入法”；
- N2 控件：设置文本框中输入文本的方式为“微软拼音输入法”；
- N3 控件：设置文本框中输入文本的方式为“郑码输入法”；
- N4 控件：设置文本框中输入文本的方式为“英文输入法”；

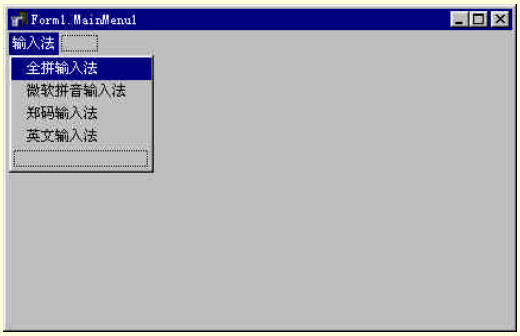


图 3-23　主菜单系统

添加控件后的主窗体如图 3-24 所示。

Green Apple Data Center

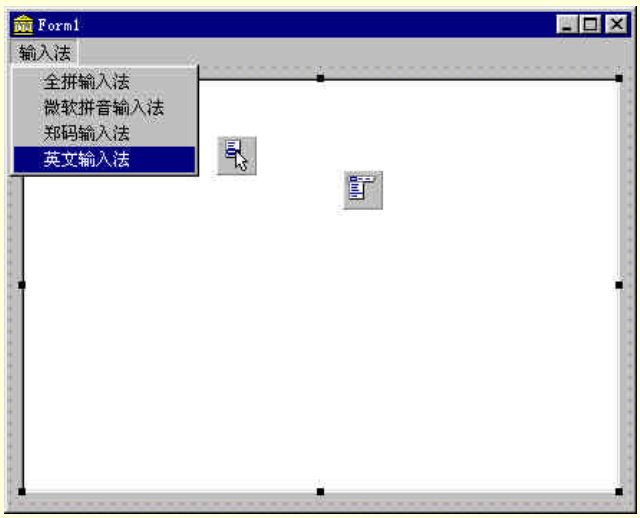


图 3-24　添加控件后的窗体

窗体中 Memo 控件和主菜单项的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 441
  Height = 354
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  Menu = MainMenu1
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Memo1: TMemo
    Left = 8
    Top = 8
    Width = 417
    Height = 289
    Lines.Strings = ('Memo1')
    TabOrder = 0
  end
  object PopupMenu1: TPopupMenu
    Left = 144
    Top = 48
    object Cut1: TMenuItem
      Caption = 'Cut'
    end
    object Copy1: TMenuItem
```

Green Apple Data Center

```
      Caption = 'Copy'
    end
    object Paste1: TMenuItem
      Caption = 'Paste'
    end
    object Delete1: TMenuItem
      Caption = 'Delete'
    end
    object Selectall1: TMenuItem
      Caption = 'Select all'
    end
  end
  object MainMenu1: TMainMenu
    Left = 232
    Top = 72
    object Style1: TMenuItem
      Caption = '输入法'
      object N1: TMenuItem
        Caption = '全拼输入法'
      end
      object N2: TMenuItem
        Caption = '微软拼音输入法'
      end
      object N3: TMenuItem
        Caption = '郑码输入法'
      end
      object N4: TMenuItem
        Caption = '英文输入法'
      end
    end
  end
end
```

2．程序的初始化

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，在缺省的情况下，代码窗口中显示的是窗体的 FormCreate()事件，将光标移动代码窗口中，并且添加下列代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
memo1.Clear;
//清空文本框中的内容
memo1.PopupMenu:=form1.PopupMenu1;
//将弹出式菜单与 Memo 控件相关联
end;
```

窗体的 FormCreate()事件在窗体的装入的同时就会被激活，进入到窗体的 FormCreate()事件的过程中，首先通过语句 memo1.Clear; 清空文本框中的内容，然后将弹出式菜单与 Memo 控件相关联。

3．响应弹出菜单事件

在弹出式菜单中有五个选项，它们所能够实现的功能如下所示：

Green Apple Data Center

- Cut：将当前所选中的数据剪切到剪贴板上；
- Copy：将当前选中的数据复制到剪贴板上；
- Paste：将当前剪贴板上的数据粘贴到当前的位置上；
- Delete：删除当前选中的数据，在剪贴板上没有复制的数据；
- Select：选中 Memo 控件上的所有数据。

在程序的设计过程中，用鼠标的左键双击如图 3-25 所示的弹出式菜单设计器中的各个子菜单项，就会弹出相应的代码窗口，把光标移动到事件处理的过程中，就可以添加相应的菜单相应代码。

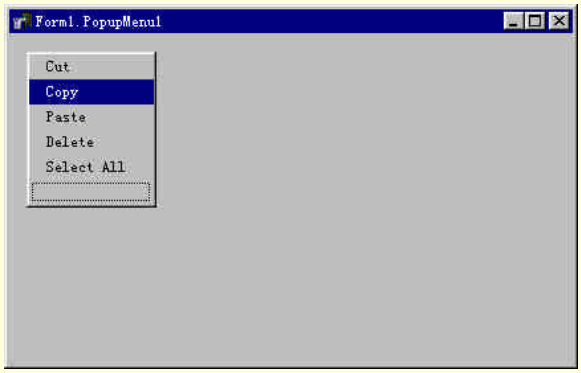


图 3-25　弹出式菜单设计器

“Cut”子菜单的响应代码如下所示：

```
procedure TForm1.Cut1Click(Sender: TObject);
begin
memo1.CutToClipboard;
//将当前选中的数据剪切到剪贴板上
end;
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击“Cut”子菜单项，就会激活 Cut1Click()事件，然后程序通过 Memo 控件的 CutToClipboard 方法把当前选中的数据剪切到剪贴板上。

“Copy”子菜单的响应代码如下所示：

```
procedure TForm1.Copy1Click(Sender: TObject);
begin
memo1.CopyToClipboard;
//将当前选中数据复制到剪贴板上
end;
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击“Copy”子菜单项，就会激活 Copy1Click()事件，然后程序通过 Memo 控件的 CopyToClipboard 方法把当前选中的数据复制到剪贴板上。

“Paste”子菜单的响应代码如下所示：

```
procedure TForm1.Paste1Click(Sender: TObject);
begin
memo1.PasteFromClipboard;
//将剪贴板上的数据放置到当前位置上
end;
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击“Paste”子菜单项，就会激活 Paste1Click()事件，然后程序通过 Memo 控件的 PasteFromClipboard 方法把当前剪贴板上的数据粘贴到控件的当前位置上。

“Delete”子菜单的响应代码如下所示：

Green Apple Data Center

```
procedure TForm1.Delete1Click(Sender: TObject);
begin
memo1.ClearSelection;
//将当前选中的数据清除
end;
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击“ Delete **”子菜单项，就会激活** Delete1Click()**事件，然后程序通过** Memo **控件的** ClearSelection **方法把当前控件上选中的数据清除。**

“ SelectAll **”子菜单的响应代码如下所示：**

```
procedure TForm1.SelectAll1Click(Sender: TObject);
begin
memo1.SelectAll;
//全选控件中的数据
end;
```

程序说明：

在程序运行的过程中，当用户用鼠标的左键单击“ SelectAll **”子菜单项，就会激活** SelectAll1Click()**事件，然后程序通过** Memo **控件的** SelectAl **方法把当前控件上的所有数据都选中。**

4．响应主菜单事件

在窗体主菜单的四个菜单项的作用是为 Memo **控件选择输入数据的文字输入法，即英文输入法、全拼输入法、郑码输入法还是微软拼音输入法。**

注意：

- **为** Memo **控件设置输入法，从实质上来讲，就是设置** Memo **控件的** ImeName **属性的属性值。**

按照同样的方法，在弹出的按钮响应代码窗口中，找到相应的相应代码，并且在它的事件响应过程中添加如下代码：

```
procedure TForm1.N1Click(Sender: TObject);
begin
   memo1.ImeName:=N1.Caption;
//设置全拼输入法
end;
procedure TForm1.N2Click(Sender: TObject);
begin
memo1.ImeName:=N2.Caption;
//设置微软拼音输入法
end;
procedure TForm1.N3Click(Sender: TObject);
begin
memo1.ImeName:=N3.Caption;
//设置郑码输入法
end;
procedure TForm1.N4Click(Sender: TObject);
begin
memo1.ImeName:=' ';
//设置英文输入法
end;
```

Green Apple Data Center

3.3.3 运行结果和参考程序

做完以上各步的工作后，选择菜单“File”中的“Save All”选项，保存文件，然后按键盘上的功能键 F9 运行程序，程序运行结果如图 3-26 所示。在程序运行的初始画面中，用鼠标的左键单击“输入法”菜单选择一项，然后输入一段文本。

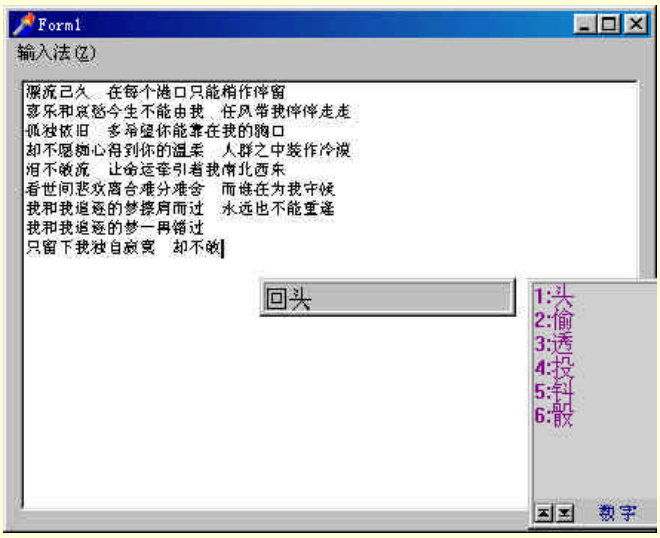


图 3-26　输入一段文本

在文本输入框中可以输入英文和汉字，在程序运行的过程中用户还可以选择文本的输入方法，在文本框上单击鼠标的右键，就会弹出一个弹出式菜单，在其中可以完成剪切、复制等功能，如图 3-27 所示。

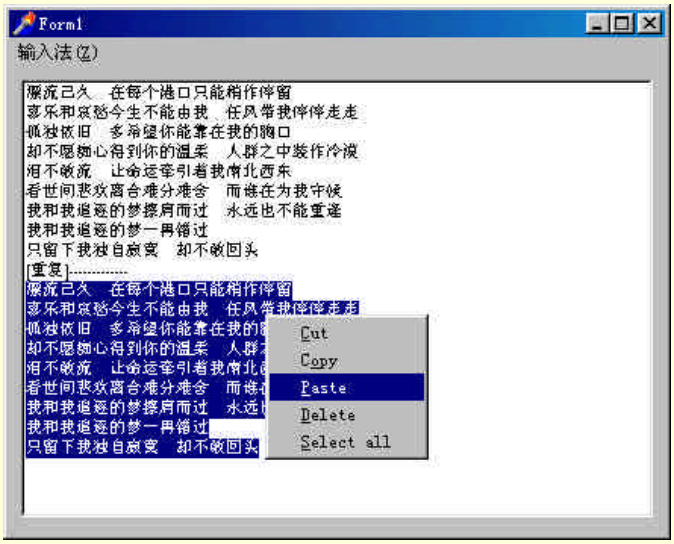


图 3-27　程序中的弹出式菜单

附源程序代码如下所示，供读者参考学习：

程序清单

```
unit popmenu;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Menus;
```

Green Apple Data Center

```
type
  TForm1 = class(TForm)
    PopupMenu1: TPopupMenu;
    Cut1: TMenuItem;
    Copy1: TMenuItem;
    Paste1: TMenuItem;
    Delete1: TMenuItem;
    Selectall1: TMenuItem;
    MainMenu1: TMainMenu;
    Style1: TMenuItem;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    Memo1: TMemo;
    procedure FormCreate(Sender: TObject);
    procedure Cut1Click(Sender: TObject);
    procedure Copy1Click(Sender: TObject);
    procedure Paste1Click(Sender: TObject);
    procedure Delete1Click(Sender: TObject);
    procedure Selectall1Click(Sender: TObject);
    procedure N1Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
memo1.Clear;
//清空文本框中的内容
memo1.PopupMenu:=form1.PopupMenu1;
//将弹出式菜单与 Memo 控件相关联
end;
```

Green Apple Data Center

```
procedure TForm1.Cut1Click(Sender: TObject);
begin
memo1.CutToClipboard;
//将当前选中的数据剪切到剪贴板上
end;

procedure TForm1.Copy1Click(Sender: TObject);
begin
memo1.CopyToClipboard;
//将当前选中数据复制到剪贴板上
end;

procedure TForm1.Paste1Click(Sender: TObject);
begin
memo1.PasteFromClipboard;
//将剪贴板上的数据放置到当前位置上
end;

procedure TForm1.Delete1Click(Sender: TObject);
begin
memo1.ClearSelection;
//将当前选中的数据清除
end;

procedure TForm1.Selectall1Click(Sender: TObject);
begin
memo1.SelectAll;
//全选控件中的数据
end;

procedure TForm1.N1Click(Sender: TObject);
begin
   memo1.ImeName:=N1.Caption;
//设置全拼输入法
end;

procedure TForm1.N2Click(Sender: TObject);
begin
   memo1.ImeName:=N2.Caption;
//设置微软拼音输入法
end;

procedure TForm1.N3Click(Sender: TObject);
begin
memo1.ImeName:=N3.Caption;
```

Green Apple Data Center

```
//设置郑码输入法
end;

procedure TForm1.N4Click(Sender: TObject);
begin
memo1.ImeName:='';
//设置英文输入法
end;
end.
```

3.4 小 结

在本章中，作者系统的介绍了程序运行过程中，人机交互的另一种常用的手段——菜单的程序设计方法。结合前一章中所讲解的内容，读者在程序开发实践中，可以利用 Delphi 系统提供的菜单和对话框设置自己的应用程序。

当然也可以自己定义菜单和对话框，这一点希望在不断的开发实践中自己加以体会。

Green Apple Data Center

第四章　Delphi 中的图形绘制

在面向对象的 Windows 应用程序中，图形的使用是非常普遍的，因为 Windows 本身就是一个庞大的图形用户界面。在 Delphi 的应用程序中，图形的绘制和处理一直是一个比较活跃的领域，任何一个应用程序中都免不了要同图形打交道，而且图形绘制又是进行多媒体程序设计的基础。

所以在本书中，我们用专门的一章讲解图形的绘制编程技巧。下面就通过几个示例程序来说明在 Delphi 中如何进行图形绘制处理。

4.1　基本图形的绘制

利用 Delphi 中提供的绘图函数可以绘制出简单的基本图案（如直线、圆和矩形等），同时也可以绘制出复杂的图案。下面我们就先对基本图案的绘制方法做以下说明。

4.1.1　直线的绘制

在 Delphi 中绘制直线是通过 LineTo 方法来实现的，它的语法结构如下所示：

```
procedure LineTo(X, Y: Integer);
```

下面通过一个示例来说明如何在程序中绘制直线，在程序运行的过程中，当用户在窗体上单击鼠标的左键时，程序就会调用相应的方法在窗体上绘制一条由上一个鼠标单击点到当前鼠标单击点的直线，步骤如下：

1．开始工作

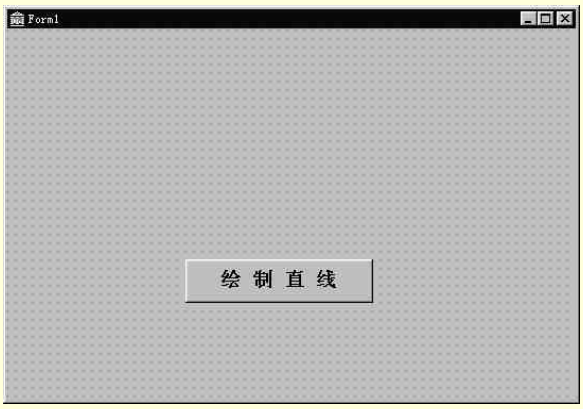


图 4-1　添加控件后的窗体

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Standart ”选项后，在“ Button ”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件，添加控件后的窗体如图 4-1 所示。

2．添加代码

在程序的设计过程中，用鼠标的左键双击窗体上的按钮控件，在屏幕上就会弹出一个代码窗口，把光标移动到控件和窗体的相应事件过程中，并且添加如下的程序代码：

程序清单

```
unit Unit1;
```

Green Apple Data Center

```
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
type
    TForm1 = class(TForm)
    Button1: TButton;
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  start:boolean;

implementation
{$R *.DFM}

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
if start then
form1.Canvas.LineTo(x,y);


//绘制直线到鼠标的单击点
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
start:=false;
//初始化变量
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
start:=true;
//设置变量值
end;
```

Green Apple Data Center

```
end.
```

程序说明：

在程序的变量声明段中，首先定义了一个布尔型变量 Start，**它的作用是控制程序绘制直线的开始时间，在窗体的初始化过程中通过语句** start:=false;**设置为** False，**如果在程序运行的过程中，用户单击按钮“绘制直线”，那么程序就自动的设置变量** Start **的值为** True，**然后通过语句** form1.Canvas.LineTo(x,y);**在窗体上绘制直线。**

3．运行程序

做完以上的工作后，按键盘上的功能键 F9 **运行程序，在程序运行的初始画面中，首先单击按钮“绘制直线”，激活绘图程序后，用鼠标的左键在窗体上单击，程序的运行的结果如图** 4-2 **所示。**

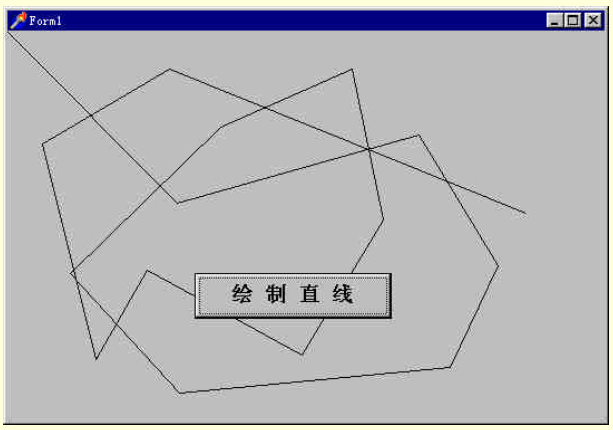


图 4-2　**程序运行结果**

4.1.2　椭圆的绘制

在 Delphi **中绘制椭圆调用的是** Ellipse **方法，它的语法结构如下所示：**

```
procedure Ellipse(X1, Y1, X2, Y2: Integer);
```

下面通过一个示例来说明如何在程序中绘制椭圆，在程序运行的过程中，当用户在窗体上按下并且移动鼠标时，程序就会调用相应的方法在窗体上绘制一个椭圆，具体的程序设计步骤如下：

1．开始工作

首先启动一个新的项目，选择菜单“ File **”中的“** New　Application **”项，在** Delphi **的集成开发环境中就会弹出一个新建的窗体。**

2．添加代码

在程序的设计过程用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到到代码段中的适当位置，并且添加如下绘制绘制椭圆的代码：

程序清单

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
    procedure FormCreate(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
```

Green Apple Data Center

```
        Y: Integer);
    private
      { Private declarations }
    public
      { Public declarations }
    end;
var
    Form1: TForm1;
    start:boolean;
    //定义一个新的变量
implementation
{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
begin
start:=false;
//变量初始化
end;

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
start:=true;
//设置变量值
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
    Y: Integer);
begin
if start then
     form1.Canvas.Ellipse(x,y,20,20);
     //绘制椭圆
end;
end.
```

3．运行程序

然后按键盘上的功能键 F9 运行程序，在程序运行的初始画面中，按下 F9 键的同时移动鼠标，在窗体上就会绘制出一系列的椭圆，程序的运行结果如图 4-3 所示。

Green Apple Data Center

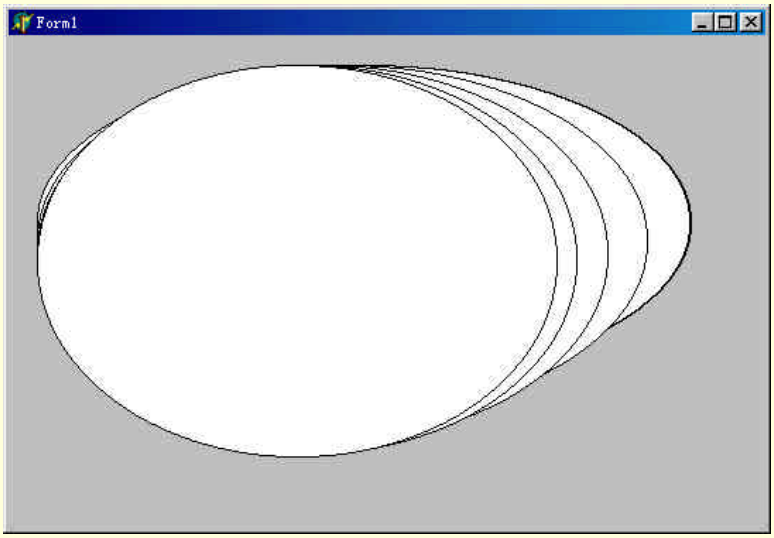


图 4-3　程序运行结果

4.1.3　绘制多边形

在 Delphi 中可以通过调用 Polyline 方法来绘制任意边数的多边形，它的语法结构如下所示：

```
procedure Polyline(Points: array of TPoint);
```

下面通过一个简单的示例来说明如何在程序中绘制任意边数的多边形，在程序运行的过程中，当用户在文本框在中输入边数后单击绘图按钮，程序就会调用相应的方法在窗体上绘制出相应的多边形。

具体的程序设计步骤如下：

1．开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项；

在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Standart ”选项后，在“ Button ”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件，然后接着向窗体上添加一个 Label 控件和一个 Edit 控件，添加控件后的窗体如图 4-4 所示。



图 4-4　添加控件后的窗体

其中各个控件的作用如下所示：

- Label 控件：显示一行固定的文本；
- Edit 控件：用于接收用户的输入，从而产生多边形的边数；
- Button 控件：产生绘制多边形的动作。

2．添加代码

在程序的设计阶段，用鼠标的左键双击窗体，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口中

Green Apple Data Center

的适当位置，并且添加如下所示的程序代码：

程序清单

```
unit Poly;
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Label1: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;

implementation
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
    count:integer;
    temp:integer;
    Polyarray:array[0..15] of tpoint;
    //定义变量
begin
    count:=strtoint(form1.edit1.text);
    if count>15 then count:=15;
    for temp:=0 to count do
    begin
    polyarray[temp]:=point(trunc(sin((2*3.1415926)*temp/count)*100)+(form1.width div 2),
    trunc(cos((2*3.1415926)*temp/count)*100)+(form1.height div 3));
    end;
    for temp:=count+1 to 15 do
    polyarray[temp]:=polyarray[0];
    form1.Canvas.Polyline(polyarray);
    //绘制多边形
```

Green Apple Data Center

```
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Canvas.Pen.Width:=4;
//设置画笔的大小
end;

end.
```

3. 运行程序

做完以上的工作后，按键盘上的功能键 F9 运行程序，在程序运行的初始画面中，在文本输入框中输入多边形的边数，然后用鼠标单击“开始绘制”按钮，程序运行结果如图 4-5 所示。

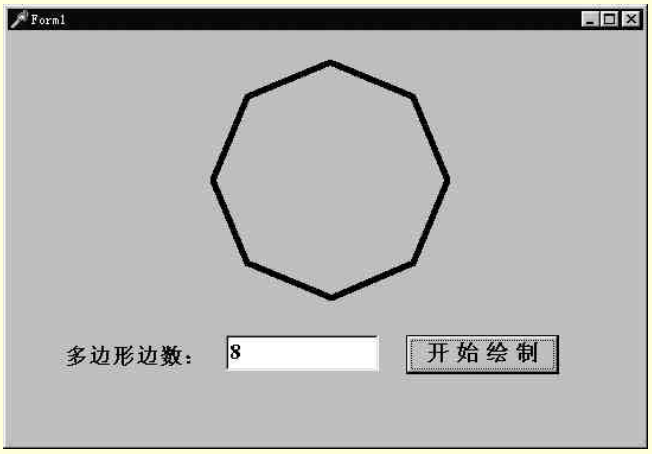


图 4-5 程序运行结果

4.1.4 绘制矩形

在 Delphi 中绘制矩形调用的方法是 Rectangle 方法，它的语法结构如下所示：

```
procedure Rectangle(X1, Y1, X2, Y2: Integer);
```

在调用 Rectangle 方法绘制矩形的时候，要为该方法提供四个顶点的坐标，并且顶点的坐标值为 Integer 类型。

下面也通过一个示例来说明如何在程序中绘制一个矩形，在程序运行的过程中，每隔 1 秒钟，在窗体上就会以随机点绘制一个矩形，具体的程序设计步骤如下：

1. 开始工作

首先启动一个新的项目，然后在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“System”选项后，在“Timer”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个计时器控件。

计时器控件的属性设置和窗体属性设置如下：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 402
  Height = 289
  Caption = 'Form1'
  Color = clGreen
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
```

Green Apple Data Center

```
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    OldCreateOrder = False
    OnCreate = FormCreate
    PixelsPerInch = 96
    TextHeight = 13
    object Timer1: TTimer
      OnTimer = Timer1Timer
      Left = 104
      Top = 40
    end
  end
```

2．添加代码

为了保证在程序运行的过程中，能够每隔 1 秒钟就绘制一个矩形，在程序的初始化过程中应该设置计时器控件的有效状态为 True，具体的程序代码如下所示：

双击记时器控件，在代码段中添加如下代码：

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
form1.Canvas.Rectangle(random(form1.width),random(form1.Height),random(form1.width),random(form1.height));
//以随机点绘制矩形
end;
```

双击窗体，在相应的代码段中添加下面的代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Timer1.Enabled:=true;
//设置控件的有效状态
end;


end.
```

程序说明：

由于是以随机点的形式绘制矩形，所以在程序运行的过程中要调用函数 random()来产生四个随机数，并且绘制矩形的代码要放在计时器控件的 Timer1Timer()过程事件中，这样才能够保证每隔 1 秒钟绘制一个随机点矩形。

3．运行程序

做完以上的工作后，按键盘上的功能键 F9 运行程序，在程序运行的初始画面中，每隔 1 秒钟，程序就会自动的以随机点来绘制一个矩形，程序运行结果如图 4-6 所示。

Green Apple Data Center

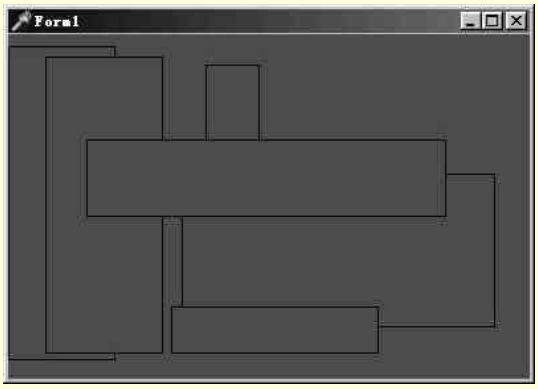


图 4-6　随机绘制的矩形

4.1.5　绘制三角曲线

在 Delphi 中可以通过调用 Lineto 方法来绘制三角曲线，包括正弦曲线、余弦曲线、正切曲线和余切曲线等。下面就通过一个绘制正弦曲线的简单示例程序向读者说明，具体的程序设计步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在窗体上添加一个标签——“正弦曲线”，然后在窗体上放置一个按钮控件和一个图片（Image）控件。

添加控件后的窗体如图 4-7 所示。



图 4-7　添加控件后的窗体

添加到窗体上控件的作用如下所示：

- 按钮“开始绘制”：执行绘制正弦曲线的程序；
- Image1 控件：作为绘制正弦曲线的容器，并且完成绘制正弦曲线的动作；

窗体和控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 356
  Height = 300
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
```

Green Apple Data Center

```
    Font.Style = []
    OldCreateOrder = False
    OnCreate = FormCreate
    PixelsPerInch = 96
    TextHeight = 13
    object Label1: TLabel
      Left = 16
      Top = 16
      Width = 84
      Height = 20
      Caption = '    正弦曲线     '
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -16
      Font.Name = 'MS Sans Serif'
      Font.Style = []
      ParentFont = False
    end
    object Image1: TImage
      Left = 16
      Top = 40
      Width = 313
      Height = 217
    end
    object BitBtn1: TBitBtn
      Left = 240
      Top = 8
      Width = 75
      Height = 25
      Caption = '开始绘制'
      TabOrder = 0
      OnClick = BitBtn1Click
    end
  end
```

2．响应窗体事件

本示例程序的作用是在窗体的图片容器上绘制出正弦曲线，为此要在程序的初始化阶段定义画笔的属性和作为绘图容器——Image 控件的显示属性。

为此，在程序的设计阶段，用鼠标左键双击窗体上的空白处，在弹出的代码窗口中选择窗体的 FormCreate 事件，并且添加如下所示的响应代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Image1.Canvas.Pen.Width:=5;
//设置画笔大小
form1.Image1.Transparent:=true;
//设置四个控件具有透明色
end;
```

Green Apple Data Center

窗体的 FormCreate 事件在程序运行初期就会被执行，程序通过 form1.Image1.Canvas.Pen.Width:=5;语句设置了画笔的大小，然后通过 form1.Image1.Transparent:=true;语句来设置四个 Image 控件在程序运行的过程中背景色都是透明的。

3．响应绘图

在程序运行的过程中，图形绘制是通过用鼠标单击“开始绘制”按钮，然后绘制的结果显示在窗体上，为此要在程序中添加如下代码：

```
procedure TForm1.BitBtn1Click(Sender: TObject);
var
    x1:real;
    y1:real;
    x2:real;
    y2:real;
    n:real;
    //定义变量
begin
x1:=0;
y1:=(form1.image1.Left+form1.Image1.Width)/4;
x2:=x1;
y2:=y1;
n:=0;
form1.Image1.Canvas.MoveTo(trunc(x2),trunc(y2));
//定义绘图起始点
while n<3.1415926*6 do
begin
x2:=x2+3.1415926/12;
y2:=y1-40*sin(n);
form1.Image1.Canvas.LineTo(trunc(x2),trunc(y2));
//绘制正弦曲线
n:=n+3.1415926/180;
end;
end;
```

程序说明：

在绘图开始阶段，首先定义了五个变量 x1、x2、y1、y2、n，其中 x1、y1 用于存储曲线绘制的起始点，而 x2、y2 则用于存储正弦曲线绘制的当前点，变量 n 用于控件曲线绘制的长度；然后程序通过语句 form1.Image1.Canvas.MoveTo(trunc(x2),trunc(y2));把当前绘图坐标移动到绘图的起始点；最后通过一个循环中的 form1.Image1.Canvas.Line To(trunc(x2),trunc(y2));语句完成绘制正弦曲线的操作。

4．运行程序

做完以上各步的工作后，选择菜单 File 中的 Save All 选项（这里我们保存项目为 P_Sin，单元文件为 USin）。然后在键盘上按功能键 F9 运行程序，在程序运行的初始画面中可以看到一个空白的窗体；但是如果单击“开始绘制”按钮，就可以绘制出相应的曲线（正弦曲线），程序运行结果如图 4-8 所示。

Green Apple Data Center

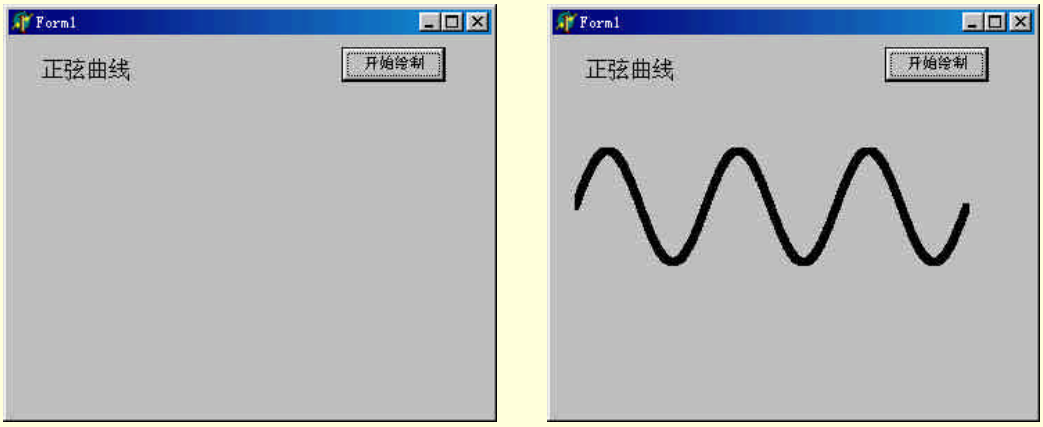


图 4-8　程序运行结果

提示 ：

根据同样的思路，用户也可以自己去联系绘制另三种三角曲线——余弦曲线（Cos）、正切曲线（Tg）和余切曲线（Ctg）。

4.2　复杂图形的设计

上节主要讲解了 Delphi 中图形各种绘制函数的使用，在本节，我们通过两个典型意义的实例来说明如何将这些基本的函数综合起来进行使用。

4.2.1　设计复杂图案

在日常的生活中，我们经常可以见到各种各样的复杂图案，如地毯上的艺术图案等，在 Delphi 中，利用一定的绘图方法，在程序中也可以绘制出丰富多彩的艺术图案，下面就以“晶体”图案为例来说明艺术图案的绘制方法。在示例程序中，用户可以在程序运行的过程中自己选择“晶体”图案的半径和分段数等参数，具体的程序设计步骤如下所示。

1．开始工作

首先启动一个新的项目，选择菜单“File”中的“New Application”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“Standard”选项后，在“Button”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件，然后接着再向窗体上添加两个 Edit 控件、两个 Label 控件和一个按钮控件，添加控件后的窗体如图 4-9 所示。

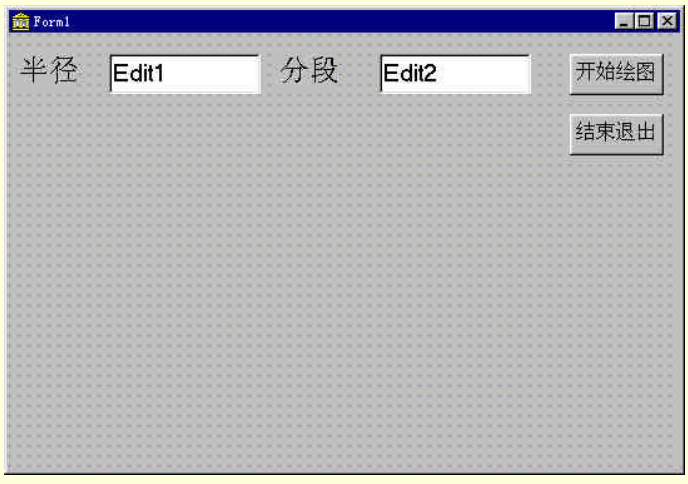


图 4-9　添加控件后的窗体

Green Apple Data Center

其中窗体上各个控件的属性设置如下所示：

```
object Form1: TForm1
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Label1: TLabel
    Left = 8
    Top = 16
    Width = 48
    Height = 29
    Caption = '半径'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -24
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
  end
  object Label2: TLabel
    Left = 216
    Top = 16
    Width = 48
    Height = 29
    Caption = '分段'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -24
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
  end
  object Button1: TButton
    Left = 448
    Top = 16
    Width = 75
    Height = 33
    Caption = '开始绘制'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
```

Green Apple Data Center

```
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 0
  end
  object Edit1: TEdit
    Left = 80
    Top = 16
    Width = 121
    Height = 32
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -19
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 1
    Text = 'Edit1'
  end
  object Edit2: TEdit
    Left = 296
    Top = 16
    Width = 121
    Height = 32
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -19
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 2
    Text = 'Edit2'
  end
  object Button2: TButton
    Left = 448
    Top = 64
    Width = 75
    Height = 33
    Caption = '结束退出'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
```

Green Apple Data Center

```
    TabOrder = 3
  end
end
```

窗体中的各个控件的作用如下所示：

- Button1 控件：产生绘制“晶体”图案的动作和完成具体的绘制工作；
- Edit1 控件：接收用户的输入，从而产生“晶体”图案的半径；
- Edit2 控件：接收用户的输入，从而产生“晶体”图案的分段数；
- Label 和 Label2 控件：显示固定的文本及提示性文字。
- Button2 控件：结束绘制“晶体”图案的动作并退出应用程序；

2．添加代码

添加代码的过程包括两个过程——程序的初始化和对按钮“开始绘图”的响应代码。

在程序的设计过程中，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到窗体 FormCreate()事件的处理过程中，并且添加如下所示的程序初始化代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Edit1.Clear;
form1.Edit2.Clear;
//清空文本框
end;
```

在程序运行的初期，系统会自动的激活窗体的 FormCreate()事件，然后程序通过语句 form1.Edit1.Clear;和 form1.Edit2.Clear;来清空窗体上文本框。

注意：

> 由于篇幅的关系，在这里对按钮“开始绘图”的响应代码就不详细的加以介绍了，读者请参看附后的源程序代码。

3．运行程序

做完以上的工作后，选择菜单“File”中的“Save All”选项，在弹出的对话框中选择合适的文件名保存文件。

然后按键盘上的功能键 F9 运行程序，在程序运行的初始画面中，在“半径”标签所对应的文本框中输入“晶体”图案的半径为 80，在“分段”标签所对应的文本框中输入“晶体”图案的分段数为 10，单击“开始绘图”按钮，程序运行结果如图 4-10 所示。

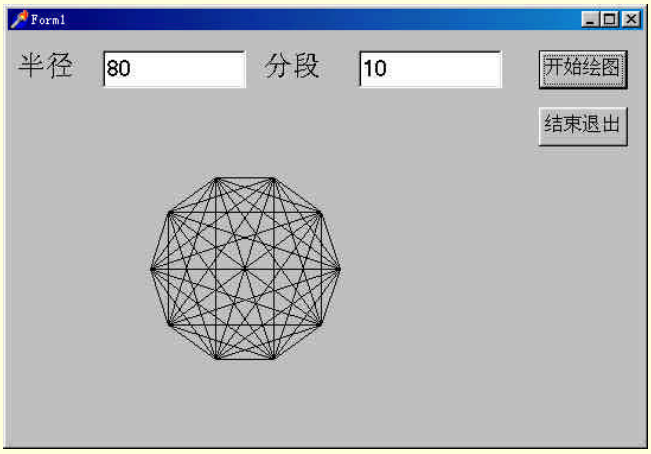


图 4-10　程序运行结果

在程序运行的过程中，用户可以随时的改变“晶体”图案的参数——半径和分段数，当我们在程序运行的过程中设置“晶体”图案的半径为 140，分段数设置为 20，程序运行结果如图 4-11 所示。

Green Apple Data Center

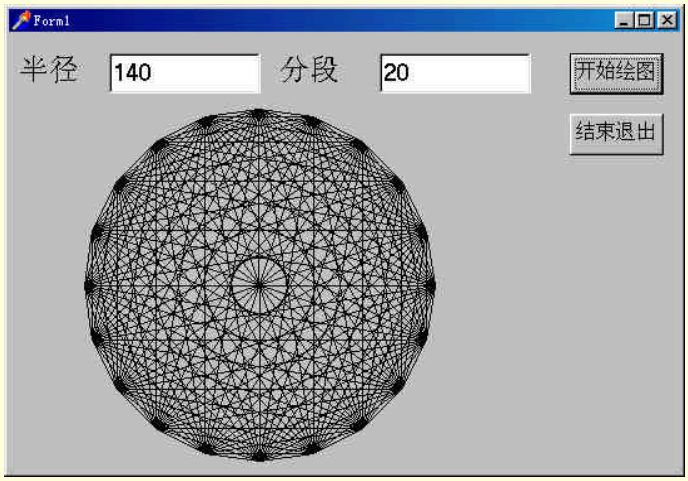


图 4-11　程序运行结果

附程序完整源代码如下所示。

程序清单

```
unit corbeil;
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Button2: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

Green Apple Data Center

```
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Edit1.Clear;
form1.Edit2.Clear;
//清空文本框
end;

procedure TForm1.Button1Click(Sender: TObject);
var t,x0,y0:real;
      r:real;
      n,j,i:integer;
      x:array[0..50] of real;
      y:array[0..50] of real;
begin
n:=strtoint(form1.Edit2.Text);
//从"分段"输入框中读取分段数
t:=6.28318/n;
r:=strtoint(form1.edit1.text);
//从"半径"输入框中读取半径
x0:= 200;
y0:= 200;
//设置图案的中心点
For i:=0 To n do
      begin
      x[i]:=r*Cos(i*t)+x0;
      y[i]:=r*Sin(i*t)+y0;
      //通过循环求出"晶体"  图案的坐标点
      end;
For i:=0 To n-2 do
            begin
            For j:=i+1 To n-1 do
                begin
                form1.Canvas.MoveTo(trunc(x[i]), trunc(y[i]));
                form1.Canvas.LineTo(trunc(x[j]), trunc(y[j]));
                //连线
                end;
            end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
close();
end;
end.
```

Green Apple Data Center

4.2.2 实时绘画程序

所谓动画图案，实际上就是绘制的图形随时变化。在日常的生活中，我们经常可以见到各种各样的动画图案，在 Delphi 中，利用一定的绘图方法，在程序中也可以绘制出小巧玲珑的动画图案，下面就以“小小时钟”为例来说明动画图案的绘制方法。

在示例程序中，用户在程序运行的过程中不需要输入任何参数，图形的动画绘制通过程序中的数学算法体现出来。具体的程序设计步骤如下所示。

1．开始工作

首先启动一个新的项目，选择菜单“File”中的“New Application”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“Standard”选项后，在“Timer”控件(记时器)的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Timer 控件，然后接着再向窗体上添加一个 Panel 控件和一个 Image 控件。添加控件后的窗体如图 4-12 所示。

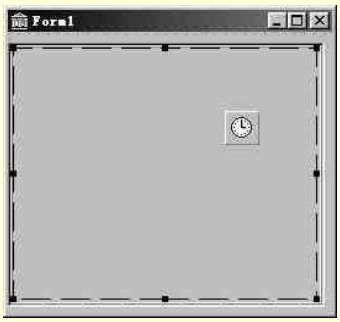


图 4-12 小小时钟的窗体界面

窗体以及响应的空间属性设置如下：

```
object Form1: TForm1
  Left = 206
  Top = 141
  BorderStyle = bsNone
  Caption = 'Form1'
  ClientHeight = 230
  ClientWidth = 263
  Color = clActiveBorder
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'System'
  Font.Style = []
  OldCreateOrder = True
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 16
  object Panel1: TPanel
    Left = 0
    Top = 8
    Width = 257
    Height = 217
    BevelInner = bvLowered
    BorderStyle = bsSingle
```

Green Apple Data Center

```
      TabOrder = 0
      object Image1: TImage
        Left = 2
        Top = 2
        Width = 247
        Height = 207
      end
    end
    object Timer1: TTimer
      OnTimer = Timer1Timer
      Left = 176
      Top = 64
    end
  end
```

窗体中各个控件的作用如下：

- Panel1 **控件：用来作为** Image1 **控件的容器；**
- Image1 **控件：图形绘制的地方，用来显示动画的时钟；**
- Timer1 **控件：用记时的方式更新绘制的图形，产生动画的效果。**

2．添加代码

添加代码的过程包括两个过程——程序的初始化和对记时器控件绘制动画时钟的响应代码。

在程序的设计过程中，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到窗体 FormCreate()**事件的处理过程中，并且添加如下所示的程序初始化代码：**

```
procedure TForm1.FormCreate(Sender: TObject);
var
  i : Integer;
begin
  with Form1.Image1.Canvas do begin
    Pen.Color := clWhite;
    Pen.Style := psSolid;
    Brush.Color := clWhite;
    Brush.Style := bsSolid;
    Rectangle(0,0,Width,Height);
//标定时钟的显示区域
    Pen.Color := clBlack;
    for i:=1 to 12 do
      Pixels[120+Round(120*Sin(i/12.0*2*Pi)),120-Round(120*Cos(i/12.0*2*Pi))] := clBlack;
//绘制时钟的圆形时刻表
  end;
  FirstTime := True;
 Form1.Panel1.Left := 0;
 Form1.Panel1.Top := 0;
 Form1.Width := Form1.Panel1.Width;
 Form1.Height := Form1.Panel1.Height;
//程序运行时将窗体设置为和时钟面板一样大
end;
```

程序说明：

Green Apple Data Center

因为时钟的时间刻度表可以是静态的，所以在窗体初始化时就可以绘制完毕，采用函数 Pixel(x,y)**完成，作用是在坐标**(x,y)**处用当前画笔绘制一个点。本段程序中用下面代码实现：**

```
for i:=1 to 12 do
Pixels[120+Round(120*Sin(i/12.0*2*Pi)),120-Round(120*Cos(i/12.0*2*Pi))] := clBlack;
```

然后，双击 Timer1 **控件，在相应的程序代码段中添加如下所示的代码：**

```
procedure TForm1.Timer1Timer(Sender: TObject);
var
  MyNow : TDateTime;
  Hour, Min,
  Sec, mSec    : Word;
  vh, vm, vs : Real;
begin
  MyNow := Now;
  DecodeTime(MyNow,Hour,Min,Sec,mSec);
//启动时间测试
  vh := Hour/12.0*2*Pi;
  vm := Min/60.0*2*Pi;
  vs := Sec/60.0*2*Pi;
  with Form1.Image1.Canvas do begin
    if not FirstTime then begin
      Pen.Color := clWhite;
      MoveTo(120,120);
      LineTo(120+Round(80*Sin(vho)),120-Round(80*Cos(vho)));
      MoveTo(120,120);
      LineTo(120+Round(100*Sin(vmo)),120-Round(100*Cos(vmo)));
      MoveTo(120,120);
      LineTo(120+Round(100*Sin(vso)),120-Round(100*Cos(vso)));
    end
    else
      FirstTime := False;
    Pen.Color := clBlack;
    MoveTo(120,120);
    LineTo(120+Round(80*Sin(vh)),120-Round(80*Cos(vh)));
    Pen.Color := clBlack;
    MoveTo(120,120);
    LineTo(120+Round(100*Sin(vm)),120-Round(100*Cos(vm)));
    Pen.Color := clRed;
    MoveTo(120,120);
    LineTo(120+Round(100*Sin(vs)),120-Round(100*Cos(vs)));
  end;
//采用更新直线来完成时针、分针和秒针的动态显示
  vho := vh;
  vmo := vm;
  vso := vs;
end;
```

Green Apple Data Center

3．运行程序

做完以上的工作后，选择菜单“ File ”中的“ Save All ”选项，在弹出的对话框中选择合适的文件名保存文件。

然后按键盘上的功能键 F9 运行程序，程序运行结果如图 4-13 所示。

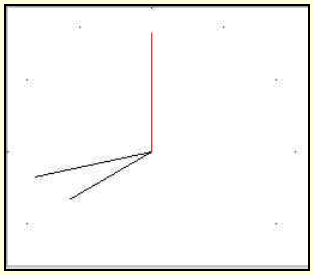


图 4-13　时钟运行画面

“ 小小时钟 ”的完整程序源代码如下，以作参考：

程序清单

```
unit Clock;

interface
uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, ExtCtrls;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Image1: TImage;
    Timer1: TTimer;
    procedure FormCreate(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
    FirstTime        : Boolean;
    vho, vmo, vso : Real;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
procedure TForm1.FormCreate(Sender: TObject);
var
```

Green Apple Data Center

```
  i : Integer;
begin
  with Form1.Image1.Canvas do begin
    Pen.Color := clWhite;
    Pen.Style := psSolid;
    Brush.Color := clWhite;
    Brush.Style := bsSolid;
    Rectangle(0,0,Width,Height);
    Pen.Color := clBlack;
    for i:=1 to 12 do
      Pixels[120+Round(120*Sin(i/12.0*2*Pi)),120-Round(120*Cos(i/12.0*2*Pi))] := clBlack;
  end;

  FirstTime := True;
 Form1.Panel1.Left := 0;
 Form1.Panel1.Top := 0;
 Form1.Width := Form1.Panel1.Width;
 Form1.Height := Form1.Panel1.Height;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var
  MyNow          : TDateTime;
  Hour, Min,
  Sec, mSec    : Word;
  vh, vm, vs : Real;
begin
  MyNow := Now;
  DecodeTime(MyNow,Hour,Min,Sec,mSec);

  vh := Hour/12.0*2*Pi;
  vm := Min/60.0*2*Pi;
  vs := Sec/60.0*2*Pi;

  with Form1.Image1.Canvas do begin
    if not FirstTime then begin
      Pen.Color := clWhite;
      MoveTo(120,120);
      LineTo(120+Round(80*Sin(vho)),120-Round(80*Cos(vho)));
      MoveTo(120,120);
      LineTo(120+Round(100*Sin(vmo)),120-Round(100*Cos(vmo)));
      MoveTo(120,120);
      LineTo(120+Round(100*Sin(vso)),120-Round(100*Cos(vso)));
    end
    else
      FirstTime := False;
```

Green Apple Data Center

```
        Pen.Color := clBlack;
        MoveTo(120,120);
        LineTo(120+Round(80*Sin(vh)),120-Round(80*Cos(vh)));
        Pen.Color := clBlack;
        MoveTo(120,120);
        LineTo(120+Round(100*Sin(vm)),120-Round(100*Cos(vm)));
        Pen.Color := clRed;
        MoveTo(120,120);
        LineTo(120+Round(100*Sin(vs)),120-Round(100*Cos(vs)));
    end;
    vho := vh;
    vmo := vm;
    vso := vs;
end;
end.
```

4.3 手工绘制图形

许多用户都使用过 Windows **的绘图程序进行图形创作，其实用** Delphi **提供的各种图形类的** ActiveX **控件，我们可以开发一个图形相关的应用程序，在这个应用程序中实现手工绘制图形。**

下面就以一个制作简易画笔应用程序的实例来说明，在此应用实例中，读者也可以学习 Delphi **中的面板** Panel **控件、加速按钮** SpeedButton **控件、图像** Image **控件和** StatusBar **控件等等的使用方法，为后面的学习打下基础。具体的程序制作步骤如下所示：**

1．开始工作

首先启动一个新的项目，选择菜单 File **中的** New Application **项，在** Delphi **的集成开发环境中就会弹出一个新建的窗体，然后向窗体上添加各种必要的控件。该项目的主窗体是** form1**，我们将它的** caption **属性设置为“简易画板”。添加控件后的窗体如图** 4-14 **所示。**

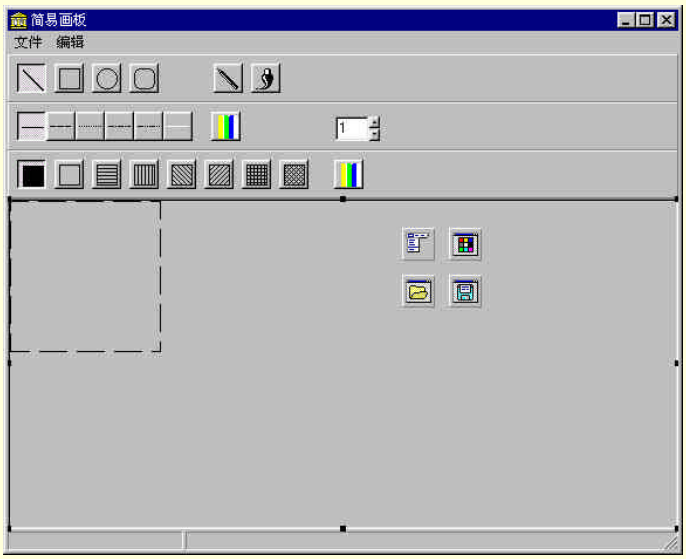


图 4-14 **添加控件后的窗体**

主窗体从上到下分为五个部分：

Green Apple Data Center

（1）菜单 MainMenu

主菜单含有“文件”和“编辑”两个主菜单。

（2）形状栏 Panel1

上面包括六个加速按钮 speedbutton，主要是选择所绘制图形的样式，包括直线（linebutton）、矩形（rectanglebutton）、椭圆（ellipsebutton）和圆角矩形（roundrectbutton）等。此外，还需要添加两个按钮用于控制是否显示画笔（penbutton）和画刷（brushbutton）。

将各个按钮的 showhint 属性设置为 True 然后为了在加速按钮上显示相应的图标，还需要设置它们的 Glyph 属性，通过调用 picture editor 对话框来设置，如图 4-15 所示。

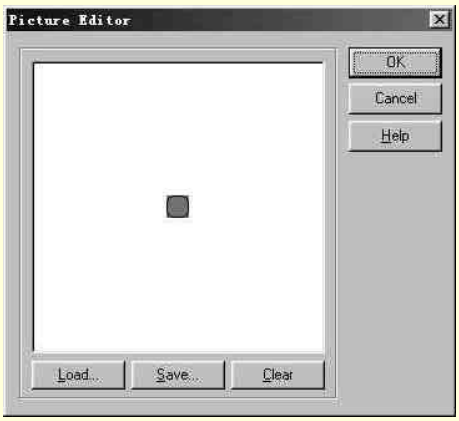


图 4-15 设置 SpeeedButton 的 Glyph 属性

设置完毕的形状栏如图 4-16 所示。



图 4-16 形状栏

（3）画笔栏

画笔栏主要用于设置画笔所用的线形以及颜色，该懒包括七个加速按钮和一个 edit 控件和一个 UpDown 控件。

七个 SpeedButton 空间如下：

- SolidPen 按钮：设置直线线形；
- DashPen 按钮：设置虚线线形；
- DotPen 按钮：设置点线线形；
- DashDotPen 按钮：设置虚点线（一个间隔点）线形；
- DashDotDotPen 按钮：设置虚点线（两个间隔点）线形；
- ClearPen 按钮：没有边线；
- PenColor 按钮：设置画笔的颜色。
- 一个 Edit 控件如下：
- PenSize 文本框：设置线宽。
- 一个 UpDown 控件如下：
- PenWidth 活动拉条：和 PenSize 控件联合使用，用来改变线宽。

设置完毕的画笔栏如图 4-17 所示。



图 4-17 画笔栏

（4）画刷栏

Green Apple Data Center

画刷栏的设置类似于画笔栏，主要用于设置绘制时的填充模式。该栏包括九个加速按钮，在设计的时候将它们的 ShowHint 属性设置为 True。九个 SpeedButton 按钮控件的功能如下：

- SolidBrush 按钮：设置实色填充
- ClearBrush 按钮：没有填充色
- HorizontalBrush 按钮：设置为水平线填充
- VerticalBrush 按钮：设置为垂直线填充
- FdigatalBrush 按钮：设置为斜线填充
- BdigitalBrush 按钮：设置为反斜线填充
- CrossBrush 按钮：设置为直线网格填充
- DiagCrossBrush 按钮：设置为对角线网格填充
- BrushColor 按钮：设置画刷的颜色

画刷栏设置完毕如图 4-18 所示。

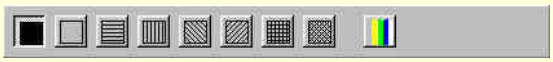


图 4-18　画刷栏

（5）图形绘制区域

窗体的下面部分便是图形绘制区域了，该区域有一个滚动文本框（ScrollBox1），其作用是当图像过大的时候进行滚动观察全貌。还有一个图像控件 Image1，程序运行的时候图形都放在该控件的画布上。

另外，窗体的底部还有一个状态栏控件，用来显示鼠标的开始位置和当前位置。

2．变量声明

在进行菜单设计之前，需要为本程序声明几个公用变量，代码如下所示：

```
  public
    { 公用变量描述 }
    BrushStyle: TBrushStyle;
//存放画刷样式
    PenStyle: TPenStyle;
//存放画笔样式
    PenWide: Integer;
//存放画笔宽度
    Drawing: Boolean;
//Bool 型变量，用来判断是否有绘图操作
    Origin, MovePt: TPoint;
//存放所绘制图形的开始点和结束点
    DrawingTool: TDrawingTool;
//存放要绘制的形状，比如直线、矩形等。
    CurrentFile: string;
//存放当前的文件名
    procedure SaveStyles;
//保存画笔和画刷的样式
    procedure RestoreStyles;
//恢复画笔和画刷的样式
    procedure DrawShape(TopLeft, BottomRight: TPoint; AMode: TPenMode);
//绘图过程
  end;
```

Green Apple Data Center

3. “文件”菜单设计

现在开始 File 菜单设计：

（1）“新建文件”菜单项（New）

新建菜单项需要弹出一个对话框，在其中输入图像的宽度和高度，如图 4-19 所示。然后按照输入的宽度和高度新建一个空白图像，并放入 Image1 控件中。

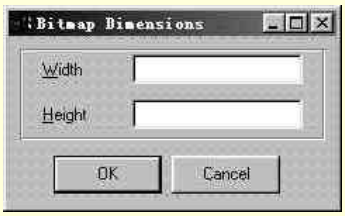


图 4-19　定义宽度和高度的对话框

该菜单项的代码如下：

```
procedure TForm1.New1Click(Sender: TObject);
var
  Bitmap: TBitmap;
begin
  with NewBMPForm do
  begin
//读取原来图像的高度和宽度
    ActiveControl := WidthEdit;
    WidthEdit.Text := IntToStr(Image.Picture.Graphic.Width);
    HeightEdit.Text := IntToStr(Image.Picture.Graphic.Height);
    if ShowModal <> idCancel then
    begin
//新建一个位图，并放入 Image1 中
      Bitmap := TBitmap.Create;
      Bitmap.Width := StrToInt(WidthEdit.Text);
      Bitmap.Height := StrToInt(HeightEdit.Text);
      SaveStyles;
      Image.Picture.Graphic := Bitmap;
      RestoreStyles;
      CurrentFile := EmptyStr;
    end;
  end;
end;
```

设置图像宽度和高度对话框的程序单元代码如下：

程序清单

```
unit BMPDlg;

interface

uses Windows, Classes, Graphics, Forms, Controls, Buttons,
  StdCtrls, ExtCtrls;
```

Green Apple Data Center

```
type
  TNewBMPForm = class(TForm)
//派生一个新的类
    OKBtn: TButton;
    CancelBtn: TButton;
    Bevel1: TBevel;
    Label1: TLabel;
    WidthEdit: TEdit;
    Label2: TLabel;
    HeightEdit: TEdit;
//定义对话框上的各个控件
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  NewBMPForm: TNewBMPForm;

implementation
{$R *.DFM}
end.
```

（2）“ 打开文件 ” 菜单项（Open）

打开菜单项调用系统提供的打开文件对话框(OpenFileDialog)控件，然后调用 LoadFromFile 函数将选择的位图文件放入到 Image1 控件中，其代码如下：

```
procedure TForm1.Open1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    CurrentFile := OpenDialog1.FileName;
    SaveStyles;
    Image.Picture.LoadFromFile(CurrentFile);
    RestoreStyles;
  end;
end;
```

（3）“ 保存 ” 菜单项（Save）

如果当前文件名不为空，则调用 SaveToFile 函数将 Image1 中的图像保存到文件中，否则便是另存为过程，代码如下：

```
procedure TForm1.Save1Click(Sender: TObject);
begin
  if CurrentFile <> EmptyStr then
    Image.Picture.SaveToFile(CurrentFile)
  else SaveAs1Click(Sender);
end;
```

（4）“ 另存为 ” 菜单项（SaveAs）

代码如下：

Green Apple Data Center

```
procedure TForm1.Saveas1Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
  begin
    CurrentFile := SaveDialog1.FileName;
    Save1Click(Sender);
  end;
end;
```

（5）“退出”菜单项（Exit）

作用是关闭窗体，代码如下：

```
procedure TForm1.Exit1Click(Sender: TObject);
begin
  Close;
end;
```

4. “编辑”主菜单设计

然后我们需要设计的是 Edit 菜单：

（1）“剪切”菜单项（Cut）

剪切分为两步，首先把图像复制到剪贴板，然后再进行清除。这里使用到了画布的 CopyRect 方法，其定义如下：

```
CopyRect(DestL:TRect, Canvas:TCanvas, Source:TRect);
```

该方法将画布上的一部分图像复制到另一部分，复制之前可以选择模式，这是由画布的 CopyMode 属性决定的。其缺省值为 cmSrcCopy，表示原样复制位图；如果在复制之前将其设置为 cmWhiteness，则用白色填充目的区域，这就相当于将图像清除了。

该菜单项的程序代码如下：

```
procedure TForm1.Cut1Click(Sender: TObject);
var
  ARect: TRect;
begin
  Copy1Click(Sender);
  with Image.Canvas do
  begin
    CopyMode := cmWhiteness;
    ARect := Rect(0, 0, Image.Width, Image.Height);
    CopyRect(ARect, Image.Canvas, ARect);
    CopyMode := cmSrcCopy;
  end;
end;
```

（2）“复制”菜单项（Copy）

复制菜单项的作用只是将 Image1 控件中的图像用剪贴板的 Assign 方法复制到剪贴板上，代码如下：

```
procedure TForm1.Copy1Click(Sender: TObject);
begin
  Clipboard.Assign(Image.Picture);
end;
```

（3）“粘贴”菜单项（Paste）

在粘贴之前首先调用剪贴板的 HastFormat 方法来判断剪贴板上是否有位图图像存在，如果有，则新建一个图像，把剪贴板上的图像传给它，然后调用画布的 Draw 方法重新画到 Image1 控件的画布上。

Green Apple Data Center

该菜单项程序代码如下：

```
procedure TForm1.Paste1Click(Sender: TObject);
var
  Bitmap: TBitmap;
begin
  if Clipboard.HasFormat(CF_BITMAP) then
  begin
    Bitmap := TBitmap.Create;
    try
      Bitmap.Assign(Clipboard);
      Image.Canvas.Draw(0, 0, Bitmap);
    finally
      Bitmap.Free;
    end;
  end;
end;
```

5．设置形状

本程序只能绘制四种图形：直线、矩形、椭圆和圆角矩形。

在定义的公共变量中，我们定义了一个 DrawingTool **变量，它主要用来存放用户所选择的图形，类型是** TdrawingTool**，这是一个枚举类型，定义如下：**

```
type
TdrawingTool=(dtLine,dtRectangle,dtEllipse,dtRoundRect);
```

因此当用户单击直线、矩形、椭圆和圆角矩形这四个加速按钮的时候，只需要将各个图形的对应值赋给 DrawingTool **变量即可。**

在绘图的时候如何使用该变量我们稍后再介绍，先将这四个按钮对应的程序代码写出：

```
procedure TForm1.LineButtonClick(Sender: TObject);
begin
  DrawingTool := dtLine;
end;

procedure TForm1.RectangleButtonClick(Sender: TObject);
begin
  DrawingTool := dtRectangle;
end;
procedure TForm1.EllipseButtonClick(Sender: TObject);
begin
  DrawingTool := dtEllipse;
end;

procedure TForm1.RoundRectButtonClick(Sender: TObject);
begin
  DrawingTool := dtRoundRect;
end;
```

6．设置画笔

画笔栏在程序开始运行的时候是隐藏的，按下形状栏中的画笔按钮才会出现，因此画笔按钮对应的代码如下：

Green Apple Data Center

```
procedure TForm1.PenButtonClick(Sender: TObject);
begin
  PenBar.Visible := PenButton.Down;
end;
```

画笔栏中控制画笔样式的按钮共有六个，她们公用一个 SetPenStyle **过程，代码如下所示：**

```
procedure TForm1.SetPenStyle(Sender: TObject);
begin
  with Image.Canvas.Pen do
  begin
    if Sender = SolidPen then Style := psSolid
    else if Sender = DashPen then Style := psDash
    else if Sender = DotPen then Style := psDot
    else if Sender = DashDotPen then Style := psDashDot
    else if Sender = DashDotDotPen then Style := psDashDotDot
    else if Sender = ClearPen then Style := psClear;
  end;
end;
```

程序说明：

其中 Sender **代表当前按下的按钮对象，通过参数判断当前按下的是什么按钮，相应的设置了画笔样式** Style **的值。**

画笔的颜色由画笔颜色按钮设置，它通过颜色对话框选择一种颜色，代码如下：

```
procedure TForm1.PenColorClick(Sender: TObject);
begin
  ColorDialog1.Color := Image.Canvas.Pen.Color;
  if ColorDialog1.Execute then
    Image.Canvas.Pen.Color := ColorDialog1.Color;
end;
```

画笔的宽度在 PenSize **编辑框的** Onchange **事件中设置，代码如下：**

```
procedure TForm1.PenSizeChange(Sender: TObject);
begin
  Image.Canvas.Pen.Width := PenWidth.Position;
end;
```

7．设置画刷

画刷栏开始也是隐藏的，用户单击形状栏中的画刷按钮时，即可显示出来。通过单击画刷按钮显示画刷栏的程序代码如下：

```
procedure TForm1.BrushButtonClick(Sender: TObject);
begin
  BrushBar.Visible := BrushButton.Down;
end;
```

画刷栏中控制画刷样式的加速按钮共有八个，它们也公用一个 SetBrushTyle **过程，代码如下所示：**

```
procedure TForm1.SetBrushStyle(Sender: TObject);
begin
  with Image.Canvas.Brush do
  begin
    if Sender = SolidBrush then Style := bsSolid
    else if Sender = ClearBrush then Style := bsClear
```

Green Apple Data Center

```
        else if Sender = HorizontalBrush then Style := bsHorizontal
        else if Sender = VerticalBrush then Style := bsVertical
        else if Sender = FDiagonalBrush then Style := bsFDiagonal
        else if Sender = BDiagonalBrush then Style := bsBDiagonal
        else if Sender = CrossBrush then Style := bsCross
        else if Sender = DiagCrossBrush then Style := bsDiagCross;
    end;
end;
```

程序说明：

该过程通过 Sender **参数来判断按下了哪个按钮，相应的设置画刷的样式。**

画刷的颜色由画刷颜色按钮设置，它通过颜色对话框选择一种颜色，实现代码如下：

```
procedure TForm1.BrushColorClick(Sender: TObject);
begin
    ColorDialog1.Color := Image.Canvas.Brush.Color;
    if ColorDialog1.Execute then
        Image.Canvas.Brush.Color := ColorDialog1.Color;
end;
```

8. **鼠标绘图**

本程序中，我们是单击鼠标左键然后移动鼠标来绘制图形的，因此利用鼠标操作在画布上绘图是本部分内容的关键。

（1）程序初始化

在绘图之前，首先需要作一些初始化的工作，前面虽然已经在窗体上放置 Image **控件，但是它还没有任何图片，也就是** Image **控件的** Picture.Graphic **属性还没有设置。**

为此，可以在主窗体的 OnCreate **事件中建立一个空位图，然后赋给** Image **控件的** Picture.Graphic **属性，代码如下：**

```
procedure TForm1.FormCreate(Sender: TObject);
var
    Bitmap: TBitmap;
begin
    Bitmap:= TBitmap.Create;
    Bitmap.Width := 200;
    Bitmap.Height := 200;
    Image.Picture.Graphic := Bitmap;
end;
```

程序说明：

把位图传给图片的 Picture.Graphic **属性就是把位图的所有权交给了图片对象，在它结束使用的时候会销毁位图图像。**

此外，绘图的时候，要使用 Image **控件的画布，只有这样才能将图形画到位图上。如果要使用窗体画布，则不便于对图形进行保存、复制等操作。**

（2）鼠标按下操作

使用鼠标绘图需要用到鼠标事件：OnMouseDown、OnMouseUp **和** OnMouseMove**，分别对应鼠标的左键按下动作、鼠标移动和左键弹起。**

无论用户什么时候按下鼠标键，OnMouseDown **事件都将发生在光标所指向的对象中，要响应鼠标按下的动作，只需要为该对象的** OnMouseDown **事件添加代码即可。**

在本例中，为了达到绘图的目的而使得 Image **控件的** OnMouseDown **事件应该处理以下的内容：**

- **设置绘画的标志；**

Green Apple Data Center

- 将光标移到画布上的当前位置；
- 设置绘画的起始点和结束点；
- 设置状态栏开始位置的值。

该事件实现的代码如下所示：

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Drawing := True;
//设置绘画的标志
  Image.Canvas.MoveTo(X, Y);
//移动鼠标指针
  Origin := Point(X, Y);
  MovePt := Origin;
//设置状态栏的开始位置
  StatusBar1.Panels[0].Text := Format('Origin: (%d, %d)', [X, Y]);
end;
```

（3）鼠标移动操作

当在 Image 控件中拖动鼠标的时候，就会产生 OnMouseMove 事件，为了实现绘图，需要添加如下的代码：

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  if Drawing then
  begin
    DrawShape(Origin, MovePt, pmNotXor);
    MovePt := Point(X, Y);
    DrawShape(Origin, MovePt, pmNotXor);
  end;
  StatusBar1.Panels[1].Text := Format('Current: (%d, %d)', [X, Y]);
end;
```

程序说明：

在 OnMouseMove 事件中调用了一个 DrawShape 过程，该过程是实现具体的绘图操作。它根据选择的形状以及通过参数传递过来的开始坐标点、结束坐标点和画笔模式来绘制不同的图形，其实现的代码如下：

```
procedure TForm1.DrawShape(TopLeft, BottomRight: TPoint; AMode: TPenMode);
begin
  with Image.Canvas do
  begin
    Pen.Mode := AMode;
//设置画笔的模式
    case DrawingTool of
//根据选择的工具绘制不同的图形
      dtLine:
        begin
          Image.Canvas.MoveTo(TopLeft.X, TopLeft.Y);
          Image.Canvas.LineTo(BottomRight.X, BottomRight.Y);
        end;
dtRectangle:Image.Canvas.Rectangle(TopLeft.X,TopLeft.Y,BottomRight.X,BottomRight.Y);
dtEllipse: Image.Canvas.Ellipse(Topleft.X, TopLeft.Y, BottomRight.X,  BottomRight.Y);
dtRoundRect:Image.Canvas.RoundRect(TopLeft.X,TopLeft.Y,BottomRight.X,  BottomRight.Y,  (TopLeft.X  -
```

Green Apple Data Center

```
BottomRight.X) div 2,          (TopLeft.Y - BottomRight.Y) div 2);
        end;
    end;
end;
```

（4）鼠标弹起操作

当鼠标按键弹起时（松开鼠标），就会产生一个 OnMouseUp **事件，在该事件中应绘制最终的图形，因此画笔的模式应该设置为** pmCopy**。该事件的程序代码如下：**

```
procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Drawing then
  begin
    DrawShape(Origin, Point(X, Y), pmCopy);
//绘制最终的图形
    Drawing := False;
  end;
end;
```

9. **运行程序**

做完以上各步的工作后，然后按选择菜单 File **中的** Save All **选项保存该项目（这些这里我们保存项目为** App6_2**，单元文件命名为** paintpad**），然后按键盘上的功能键** F9 **运行程序，程序运行的初始画面如图** 4-20 **所示，绘图程序中仅仅有一个空白的画布（图像）。**

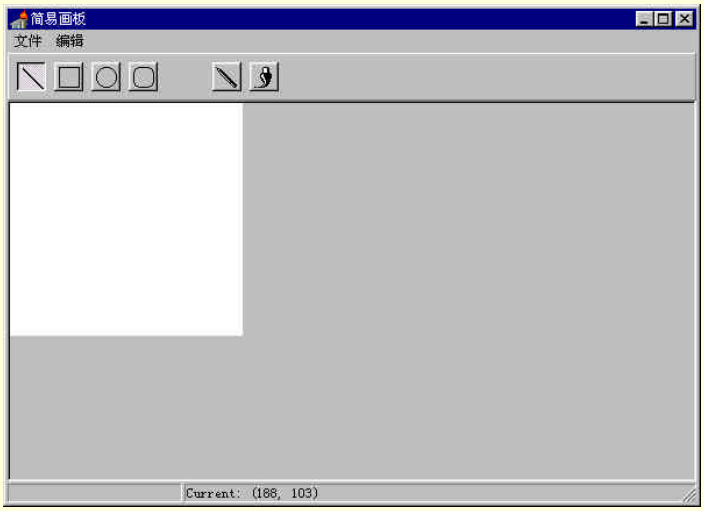


图 4-20 **绘图程序启动**

为了改变画布的大小，可以单击“ 文件 ”菜单上的“ 新文件 ”菜单项，然后弹出一个设置图像宽度和高度的对话框，如图 4-21 **所示。**

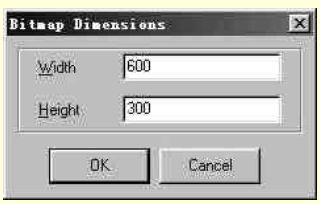


图 4-21 **设置画布（图像）的高度和宽度**

按照用户给出画布的高度（300**）和宽度（**600**）后，空白的画布就重新显示出来，如图** 4-22 **所示。**

Green Apple Data Center

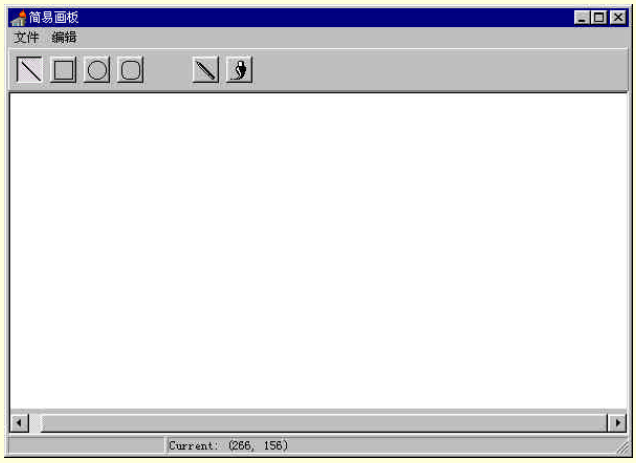


图 4-22 设置大小后的图像

然后，可以试着在画布上绘制一些不同的图形，结果如图 4-23 所示。

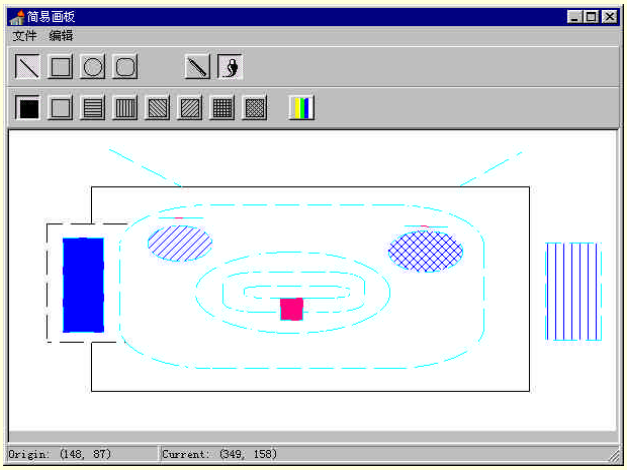


图 4-23 图形绘制结果

实现本程序的完整功能还需要添加其他的辅助代码，现附本示例的程序代码清单如下：

程序清单

```
unit paintpad;

interface

uses
  SysUtils, Windows, Messages, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, ExtCtrls, StdCtrls, ComCtrls, Menus;

type
  TDrawingTool = (dtLine, dtRectangle, dtEllipse, dtRoundRect);
  TForm1 = class(TForm)
    Panel1: TPanel;
    LineButton: TSpeedButton;
    RectangleButton: TSpeedButton;
    EllipseButton: TSpeedButton;
```

Green Apple Data Center

```
RoundRectButton: TSpeedButton;
PenButton: TSpeedButton;
BrushButton: TSpeedButton;
PenBar: TPanel;
BrushBar: TPanel;
SolidPen: TSpeedButton;
DashPen: TSpeedButton;
DotPen: TSpeedButton;
DashDotPen: TSpeedButton;
DashDotDotPen: TSpeedButton;
ClearPen: TSpeedButton;
PenWidth: TUpDown;
PenSize: TEdit;
StatusBar1: TStatusBar;
ScrollBox1: TScrollBox;
Image: TImage;
SolidBrush: TSpeedButton;
ClearBrush: TSpeedButton;
HorizontalBrush: TSpeedButton;
VerticalBrush: TSpeedButton;
FDiagonalBrush: TSpeedButton;
BDiagonalBrush: TSpeedButton;
CrossBrush: TSpeedButton;
DiagCrossBrush: TSpeedButton;
PenColor: TSpeedButton;
BrushColor: TSpeedButton;
ColorDialog1: TColorDialog;
MainMenu1: TMainMenu;
File1: TMenuItem;
New1: TMenuItem;
Open1: TMenuItem;
Save1: TMenuItem;
Saveas1: TMenuItem;
N1: TMenuItem;
Exit1: TMenuItem;
Edit1: TMenuItem;
Cut1: TMenuItem;
Copy1: TMenuItem;
Paste1: TMenuItem;
OpenDialog1: TOpenDialog;
SaveDialog1: TSaveDialog;
procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
procedure FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
```

Green Apple Data Center

```
      Y: Integer);
    procedure LineButtonClick(Sender: TObject);
    procedure RectangleButtonClick(Sender: TObject);
    procedure EllipseButtonClick(Sender: TObject);
    procedure RoundRectButtonClick(Sender: TObject);
    procedure PenButtonClick(Sender: TObject);
    procedure BrushButtonClick(Sender: TObject);
    procedure SetPenStyle(Sender: TObject);
    procedure PenSizeChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SetBrushStyle(Sender: TObject);
    procedure PenColorClick(Sender: TObject);
    procedure BrushColorClick(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure Open1Click(Sender: TObject);
    procedure Save1Click(Sender: TObject);
    procedure Saveas1Click(Sender: TObject);
    procedure New1Click(Sender: TObject);
    procedure Copy1Click(Sender: TObject);
    procedure Cut1Click(Sender: TObject);
    procedure Paste1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    BrushStyle: TBrushStyle;
    PenStyle: TPenStyle;
    PenWide: Integer;
    Drawing: Boolean;
    Origin, MovePt: TPoint;
    DrawingTool: TDrawingTool;
    CurrentFile: string;
    procedure SaveStyles;
    procedure RestoreStyles;
    procedure DrawShape(TopLeft, BottomRight: TPoint; AMode: TPenMode);
  end;

var
  Form1: TForm1;

implementation

uses BMPDlg, Clipbrd;

{$R *.DFM}
```

Green Apple Data Center

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  Drawing := True;
  Image.Canvas.MoveTo(X, Y);
  Origin := Point(X, Y);
  MovePt := Origin;
  StatusBar1.Panels[0].Text := Format('Origin: (%d, %d)', [X, Y]);
end;

procedure TForm1.FormMouseUp(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Drawing then
  begin
    DrawShape(Origin, Point(X, Y), pmCopy);
    Drawing := False;
  end;
end;

procedure TForm1.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
  Y: Integer);
begin
  if Drawing then
  begin
    DrawShape(Origin, MovePt, pmNotXor);
    MovePt := Point(X, Y);
    DrawShape(Origin, MovePt, pmNotXor);
  end;
  StatusBar1.Panels[1].Text := Format('Current: (%d, %d)', [X, Y]);
end;

procedure TForm1.LineButtonClick(Sender: TObject);
begin
  DrawingTool := dtLine;
end;

procedure TForm1.RectangleButtonClick(Sender: TObject);
begin
  DrawingTool := dtRectangle;
end;

procedure TForm1.EllipseButtonClick(Sender: TObject);
begin
  DrawingTool := dtEllipse;
end;
```

Green Apple Data Center

```
procedure TForm1.RoundRectButtonClick(Sender: TObject);
begin
  DrawingTool := dtRoundRect;
end;

procedure TForm1.DrawShape(TopLeft, BottomRight: TPoint; AMode: TPenMode);
begin
  with Image.Canvas do
  begin
    Pen.Mode := AMode;
    case DrawingTool of
      dtLine:
        begin
          Image.Canvas.MoveTo(TopLeft.X, TopLeft.Y);
          Image.Canvas.LineTo(BottomRight.X, BottomRight.Y);
        end;
      dtRectangle: Image.Canvas.Rectangle(TopLeft.X, TopLeft.Y, BottomRight.X,
        BottomRight.Y);
      dtEllipse: Image.Canvas.Ellipse(Topleft.X, TopLeft.Y, BottomRight.X,
        BottomRight.Y);
      dtRoundRect: Image.Canvas.RoundRect(TopLeft.X, TopLeft.Y, BottomRight.X,
        BottomRight.Y, (TopLeft.X - BottomRight.X) div 2,
        (TopLeft.Y - BottomRight.Y) div 2);
    end;
  end;
end;

procedure TForm1.PenButtonClick(Sender: TObject);
begin
  PenBar.Visible := PenButton.Down;
end;

procedure TForm1.BrushButtonClick(Sender: TObject);
begin
  BrushBar.Visible := BrushButton.Down;
end;

procedure TForm1.SetPenStyle(Sender: TObject);
begin
  with Image.Canvas.Pen do
  begin
    if Sender = SolidPen then Style := psSolid
    else if Sender = DashPen then Style := psDash
    else if Sender = DotPen then Style := psDot
    else if Sender = DashDotPen then Style := psDashDot
```

Green Apple Data Center

```
      else if Sender = DashDotDotPen then Style := psDashDotDot
      else if Sender = ClearPen then Style := psClear;
    end;
end;

procedure TForm1.PenSizeChange(Sender: TObject);
begin
    Image.Canvas.Pen.Width := PenWidth.Position;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
    Bitmap: TBitmap;
begin
    Bitmap := TBitmap.Create;
    Bitmap.Width := 200;
    Bitmap.Height := 200;
    Image.Picture.Graphic := Bitmap;
end;

procedure TForm1.SetBrushStyle(Sender: TObject);
begin
    with Image.Canvas.Brush do
    begin
      if Sender = SolidBrush then Style := bsSolid
      else if Sender = ClearBrush then Style := bsClear
      else if Sender = HorizontalBrush then Style := bsHorizontal
      else if Sender = VerticalBrush then Style := bsVertical
      else if Sender = FDiagonalBrush then Style := bsFDiagonal
      else if Sender = BDiagonalBrush then Style := bsBDiagonal
      else if Sender = CrossBrush then Style := bsCross
      else if Sender = DiagCrossBrush then Style := bsDiagCross;
    end;
end;

procedure TForm1.PenColorClick(Sender: TObject);
begin
    ColorDialog1.Color := Image.Canvas.Pen.Color;
    if ColorDialog1.Execute then
      Image.Canvas.Pen.Color := ColorDialog1.Color;
end;

procedure TForm1.BrushColorClick(Sender: TObject);
begin
    ColorDialog1.Color := Image.Canvas.Brush.Color;
    if ColorDialog1.Execute then
```

Green Apple Data Center

```
    Image.Canvas.Brush.Color := ColorDialog1.Color;
end;

procedure TForm1.Exit1Click(Sender: TObject);
begin
  Close;
end;

procedure TForm1.Open1Click(Sender: TObject);
begin
  if OpenDialog1.Execute then
  begin
    CurrentFile := OpenDialog1.FileName;
    SaveStyles;
    Image.Picture.LoadFromFile(CurrentFile);
    RestoreStyles;
  end;
end;

procedure TForm1.Save1Click(Sender: TObject);
begin
  if CurrentFile <> EmptyStr then
    Image.Picture.SaveToFile(CurrentFile)
  else SaveAs1Click(Sender);
end;

procedure TForm1.Saveas1Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
  begin
    CurrentFile := SaveDialog1.FileName;
    Save1Click(Sender);
  end;
end;


procedure TForm1.New1Click(Sender: TObject);
var
  Bitmap: TBitmap;
begin
  with NewBMPForm do
  begin
    ActiveControl := WidthEdit;
    WidthEdit.Text := IntToStr(Image.Picture.Graphic.Width);
    HeightEdit.Text := IntToStr(Image.Picture.Graphic.Height);
    if ShowModal <> idCancel then
```

Green Apple Data Center

```
      begin
        Bitmap := TBitmap.Create;
        Bitmap.Width := StrToInt(WidthEdit.Text);
        Bitmap.Height := StrToInt(HeightEdit.Text);
        SaveStyles;
        Image.Picture.Graphic := Bitmap;
        RestoreStyles;
        CurrentFile := EmptyStr;
      end;
    end;
end;

procedure TForm1.Copy1Click(Sender: TObject);
begin
    Clipboard.Assign(Image.Picture);
end;

procedure TForm1.Cut1Click(Sender: TObject);
var
    ARect: TRect;
begin
    Copy1Click(Sender);
    with Image.Canvas do
    begin
      CopyMode := cmWhiteness;
      ARect := Rect(0, 0, Image.Width, Image.Height);
      CopyRect(ARect, Image.Canvas, ARect);
      CopyMode := cmSrcCopy;
    end;
end;

procedure TForm1.Paste1Click(Sender: TObject);
var
    Bitmap: TBitmap;
begin
    if Clipboard.HasFormat(CF_BITMAP) then
    begin
      Bitmap := TBitmap.Create;
      try
        Bitmap.Assign(Clipboard);
        Image.Canvas.Draw(0, 0, Bitmap);
      finally
        Bitmap.Free;
      end;
    end;
end;
```

Green Apple Data Center

```
procedure TForm1.SaveStyles;
begin
  with Image.Canvas do
  begin
    BrushStyle := Brush.Style;
    PenStyle := Pen.Style;
    PenWide := Pen.Width;
  end;
end;

procedure TForm1.RestoreStyles;
begin
  with Image.Canvas do
  begin
    Brush.Style := BrushStyle;
    Pen.Style := PenStyle;
    Pen.Width := PenWide;
  end;
end;
end.
```

4.4 小 结

本章系统的介绍了在 Delphi 中进行文本、图形和图像处理的各种方法，其中“ 文本编辑器 ”、“ 图像观察器 ”和“ 晶体图案 ”等示例程序都由很大的实用性，在应用程序的设计过程中，本章的程序设计方法有很大的参考价值，希望读者在理解的基础上，设计出更好的文本、图形和图像处理程序。

Green Apple Data Center

第五章　用户窗体的设计和使用

在 Delphi 中，常见的用户窗体开发包括四个方面的内容：单文档界面窗体（SDI）、多页窗体设计、复合窗体设计和多文档界面窗体设计（MDI）。

文档界面设计相对来说比较容易让人理解，就是应用程序运行的过程中在同一个时间内只能打开一个文档来进行处理；多页窗体指的是在一个窗体上同时包含多个页面，通过单击页面的标签可以让其中任意一页在窗体的最前面显示，每一页可以象设计普通窗体那样设计；复合窗体程序指的是在一个应用程序中有多个窗口界面，按照程序的要求分别的显示在屏幕上，但是要对每一个窗体进行单独的设计；多文档界面则是指在一个父窗口中包含有多个子窗口，在程序运行过程中可以同时打开多个窗口，这些窗体之间的关系由程序指定。

好了，为了更好的理解上面所讲，还是继续学习下面介绍的内容吧。

5.1　快速使用向导

在 Delphi 中系统为用户提供了许多应用程序向导（Application Wizard），利用这些向导程序可以节省很多的劳动，可以少用甚至不用添加代码就可以生成一个应用程序。

使用向导，用户可以非常轻松的得到一个标准 Windows 单文档界面窗体（SDI），下面就利用向导来生成一个简单的文档窗体应用程序，具体的程序设计步骤如下所示。

1．开始工作

启动新项目，选择菜单 File 中的 New 项，在 Delphi 的集成开发环境中就会弹出如图 5-1 所示的名为 New Items 的对话框，用户在其中可以选择各种向导程序和窗体形式等。

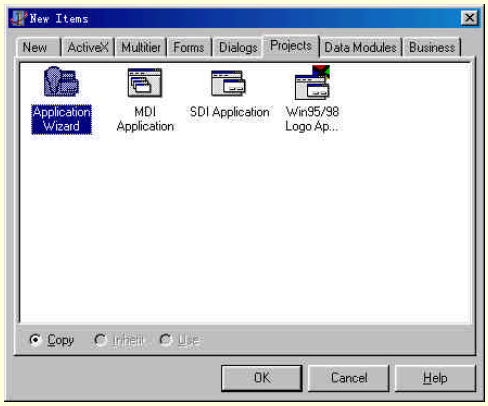


图 5-1　New Items 对话框

在 New Item 对话框中有八个选项——New、ActiveX、Multier、Forms、Dialogs、Projects、Data Modules、Business，我们选择 Projects 选项中的 Application Wizard。

2．选择菜单

单击如图 5-1 所示 New Item 对话框的 OK 按钮，然后在由向导生成的应用程序中，用户可以在如图 5-2 所示的菜单选择对话框中选择标准菜单的类型。

Green Apple Data Center

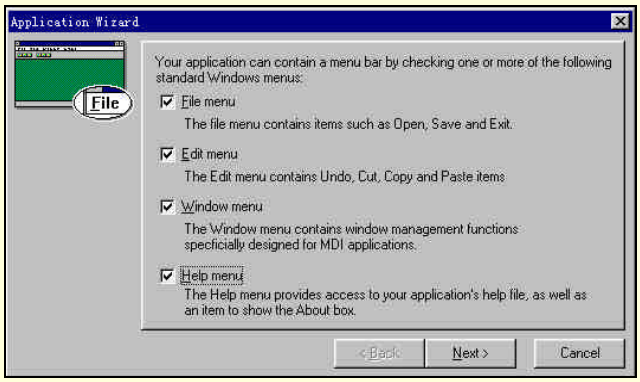


图 5-2　菜单选择窗口

在菜单选择对话框中，用户可以选择：

- File 菜单：包括打开、保存和另存为等通用菜单项；
- Edit 菜单：包括剪切、拷贝和粘贴等通用菜单项；
- Windows 菜单：包括为 MDI（多文档界面）设计的专门的管理函数菜单项；
- Help 菜单：包括索引和提示等。

在菜单选择对话框中选中四个菜单，然后用鼠标单击 Next 按钮进入下一步。

3. 选择工具栏

在应用程序中，我们经常可以看到作为菜单加速控制的工具栏，在 Delphi 的向导程序中，用户也可以对工具栏进行选择操作，如图 5-3 即为一个设计工具栏的对话框，单击 Insert 按钮即可添加相应的加速按钮。

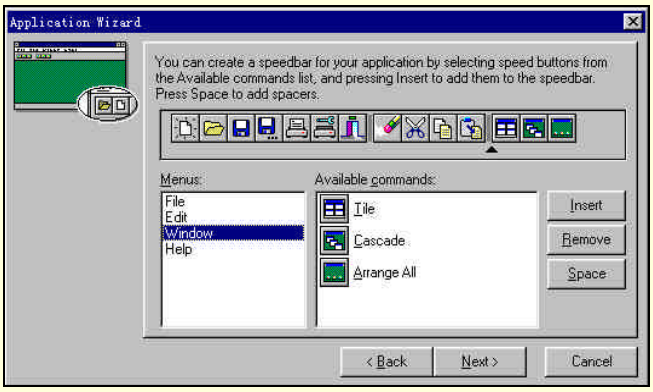


图 5-3　设计工具栏上的加速按钮

在工具栏设计对话框中用鼠标单击 Next 按钮进入下一步。

4. 存储项目

在向导程序的最后一个对话框中，用户可以为生成的应用程序选择一个有效的驱动器路径和一个应用程序的文件名，如图 5-4 所示。

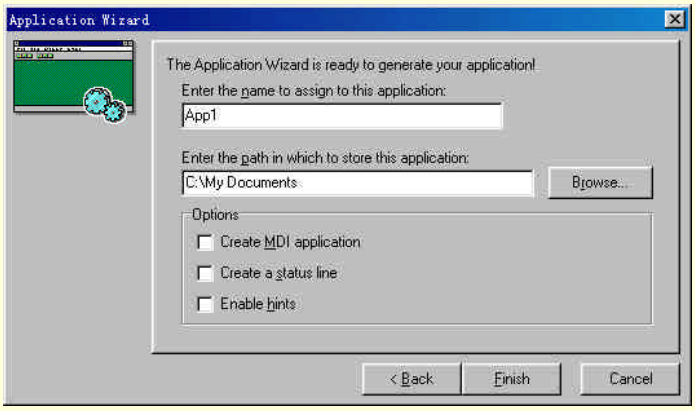


图 5-4　选择程序路径和设置

Green Apple Data Center

在这个对话框中不但可以选择程序的路径和存储文件名，还可以对程序的特性进行设置，如是否作为一个 MDI 程序，是否建立一个状态条，在状态条中是否显示状态信息等。

在对话框中选择好应用程序的路径和文件名后，在 Options 选项卡下选中三个选项，单击 Finish 按钮就完成了应用程序的设计工作。

最后，由向导设计完毕的应用程序的窗口界面如图 5-5 所示。

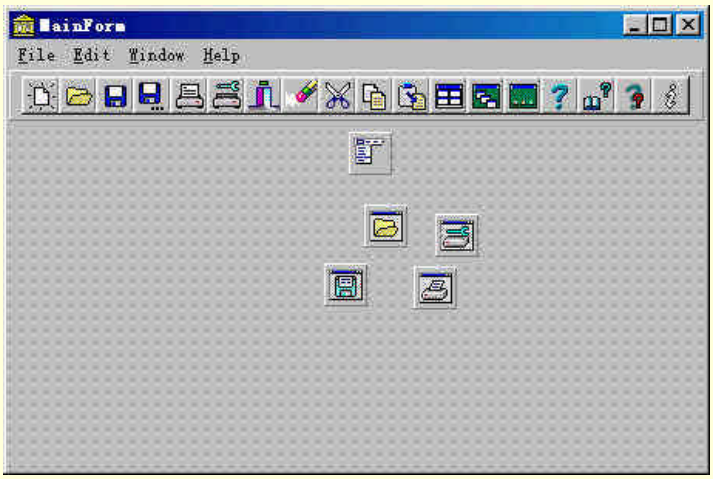


图 5-5　设计完成的工具栏

5．运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存项目文件（我们这里保存为“App1-1”），然后按键盘上的功能键 F9 运行程序，程序的运行的结果如图 5-6 所示。

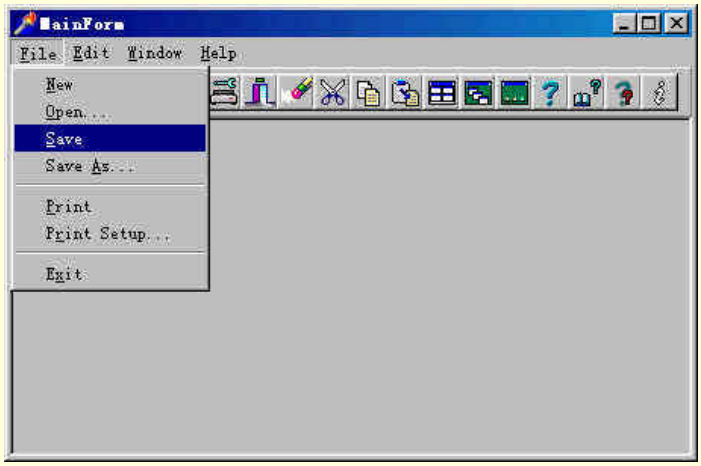


图 5-6　程序运行结果

但是在这个应用程序中，用于还没有添加各个事件的响应代码，所以它所能实现的功能还很不完善。但是向导为我们生成了一个标准的文档窗体框架，用户可以在此基础上添加自己的代码，而这正是你在学习完前面的内容后可以实现的。

5.2　多页窗体开发

所谓多页窗体就是一个窗体上有多个不同的页面，这是目前一种非常流行的图形界面风格。在每个页面上都可以象在窗体上设计用户界面，而且都可以拥有自己的控件对象这样就无形扩大了窗体的使用空间。一个典型的多页窗体就是查找和替换对话框窗体，如图 5-7 所示。

Green Apple Data Center

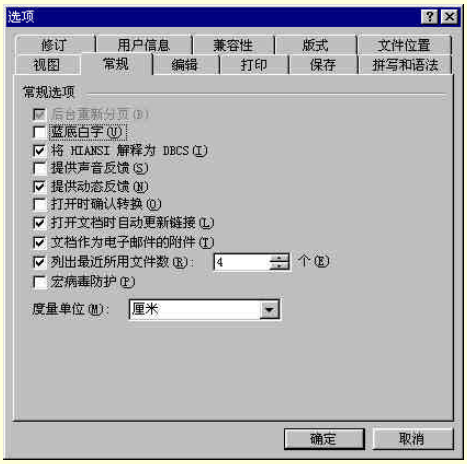


图 5-7　典型的多页窗体

在 Delphi 中，TabbedNotebook 控件可以轻松的建立一个多页窗体，经常在其上放置不同的组选项按钮，也可以为相似的控件分组。在它的设计阶段，用户可以通过它的 Pages 属性来定制 TabbedNotebook 控件的标签页，在其中可以编辑、添加、删除各个标签页，还可以改变各个标签页之间的相对位置关系。

下面就通过一个绘制正弦曲线、余弦曲线、正切曲线和余切曲线的示例程序向读者说明 TabbedNotebook 控件的使用技术，具体的程序设计步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Win3.1 选项后，在 TabbedNotebook 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 TabbedNotebook 控件。

设置 TabbedNotebook 控件的 Page 属性，单击在如图 5-8 所示的 PageEditor 对话框上为 TabbedNotebook 控件添加四个标签页的名称：“正弦曲线”、“余弦曲线”、“正切曲线”和“余切曲线”。

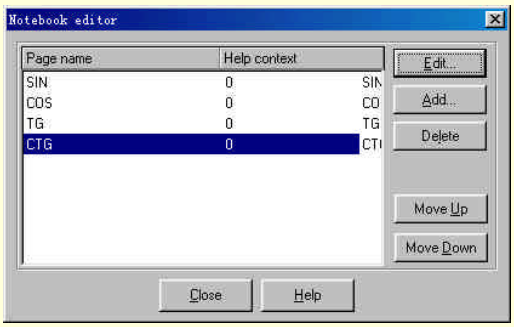


图 5-8　设置标签页面

在 TabbedNotebook 控件上添加四个标签页——“正弦曲线 SIN”、“余弦曲线 COS”、“正切曲线 TG”和“余切曲线 CTG”，并且在每个标签页上都放置一个 Image 控件，窗体如图 5-9 所示。

Green Apple Data Center



图 5-9　添加控件后的窗体

添加到窗体上控件的作用如下所示：

- **标签页“正弦曲线”：显示绘制的正弦曲线，并且隐藏其余三个标签页上的曲线；**
- **标签页“余弦曲线”：显示绘制的余弦曲线，并且隐藏其余三个标签页上的曲线；**
- **标签页“正切曲线”：显示绘制的正切曲线，并且隐藏其余三个标签页上的曲线；**
- **标签页“余切曲线”：显示绘制的余切曲线，并且隐藏其余三个标签页上的曲线；**
- Image1 **控件：作为绘制正弦曲线的容器，并且完成绘制正弦曲线的动作；**
- Image2 **控件：作为绘制余弦曲线的容器，并且完成绘制余弦曲线的动作；**
- Image3 **控件：作为绘制正切曲线的容器，并且完成绘制正切曲线的动作；**
- Image4 **控件：作为绘制余切曲线的容器，并且完成绘制余切曲线的动作。**

窗体和控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 429
  Height = 344
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object Label1: TLabel
    Left = 96
    Top = 8
    Width = 224
    Height = 37
    Caption = '多页窗体示例'
    Font.Charset = DEFAULT_CHARSET
```

Green Apple Data Center

```
    Font.Color = clWindowText
    Font.Height = -32
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
  end
  object TabbedNotebook1: TTabbedNotebook
    Left = 16
    Top = 56
    Width = 377
    Height = 250
    PageIndex = 3
    TabFont.Charset = DEFAULT_CHARSET
    TabFont.Color = clBtnText
    TabFont.Height = -11
    TabFont.Name = 'MS Sans Serif'
    TabFont.Style = []
    TabOrder = 0
    object TTabPage
      Left = 4
      Top = 24
      Caption = '正弦曲线'
      object Image1: TImage
        Left = 8
        Top = 8
        Width = 353
        Height = 209
        OnClick = Image1Click
      end
    end
    object TTabPage
      Left = 4
      Top = 24
      Caption = '余弦曲线'
      object Image2: TImage
        Left = 0
        Top = 8
        Width = 369
        Height = 209
        OnClick = Image2Click
      end
    end
    object TTabPage
      Left = 4
      Top = 24
      Caption = '正切曲线'
```

Green Apple Data Center

```
        object Image3: TImage
          Left = 8
          Top = 0
          Width = 361
          Height = 217
          OnClick = Image3Click
        end
      end
      object TTabPage
        Left = 4
        Top = 24
        Caption = '余切曲线'
        object Image4: TImage
          Left = 0
          Top = 0
          Width = 369
          Height = 217
          OnClick = Image4Click
        end
      end
    end
  end
```

2．响应窗体事件

本示例程序的作用是在不同的标签页上能够绘制出正弦、余弦等四种曲线，为此要在程序的初始化阶段定义画笔的属性和作为绘图容器——Image 控件的显示属性。

为此，在程序的设计阶段，用鼠标左键双击窗体上的空白处，在弹出的代码窗口中选择窗体的 FormCreate 事件，并且添加如下所示的响应代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Image1.Canvas.Pen.Width:=5;
form1.Image2.Canvas.Pen.Width:=5;
form1.Image3.Canvas.Pen.Width:=5;
form1.Image4.Canvas.Pen.Width:=5;
//设置画笔大小
form1.Image1.Transparent:=true;
form1.Image2.Transparent:=true;
form1.Image3.Transparent:=true;
form1.Image4.Transparent:=true;
//设置四个控件具有透明色
end;
```

窗体的 FormCreate 事件在程序运行初期就会被执行，程序通过 form1.Image1.Canvas.Pen.Width:=5;等四条语句设置了画笔的大小，然后通过 form1.Image1.Transparent:=true;等四条语句来设置四个 Image 控件在程序运行的过程中背景色都是透明的。

3．响应绘图

在程序运行的过程中，切换到不同的标签页就会隐藏其余三个标签页上的控件和图形，但是并能绘制图形，图形绘制是通过用鼠标单击 Image 控件来实现的，为此要在程序中添加如下代码（由于篇幅的关系，在这里

Green Apple Data Center

仅列举了 Image1 控件的响应代码，其余控件的代码请参看附后的程序清单）：

```
procedure TForm1.Image1Click(Sender: TObject);
var
    x1:real;
    y1:real;
    x2:real;
    y2:real;
    n:real;
    //定义变量
begin
x1:=0;
y1:=(form1.image1.Left+form1.Image1.Width)/4;
x2:=x1;
y2:=y1;
form1.Image1.Canvas.MoveTo(trunc(x2),trunc(y2));
//定义绘图起始点
n:=0;
while n<3.1415926*6 do
begin
x2:=x2+3.1415926/12;
y2:=y1-40*sin(n);
form1.Image1.Canvas.LineTo(trunc(x2),trunc(y2));
//绘制正弦曲线
n:=n+3.1415926/180;
end;
end;
```

在程序运行的过程中，切换到标签页“正弦曲线”，那么 Image1 控件就已经准备好绘制正弦曲线了，如果用户用鼠标左键单击控件 Image1，那么就会激活控件的 Image1Click 事件，开始绘图。

在绘图开始阶段，首先定义了五个变量 x1、x2、y1、y2、n，其中 x1、y1 用于存储曲线绘制的起始点，而 x2、y2 则用于存储正弦曲线绘制的当前点，变量 n 用于控件曲线绘制的长度；然后程序通过语句 form1.Image1.Canvas.MoveTo(trunc(x2),trunc(y2));把当前绘图坐标移动到绘图的起始点；最后通过一个循环中的 form1.Image1.Canvas.Line To(trunc(x2),trunc(y2));语句完成绘制正弦曲线的操作。

4．运行程序

做完以上各步的工作后，选择菜单 File 中的 Save All 选项。然后在键盘上按功能键 F9 运行程序，在程序运行的初始画面中可以看到四个空白的标签页，如果单击上面的 Image 控件，就可以绘制出相应的曲线，程序运行结果如图 5-10 所示为正弦曲线。

Green Apple Data Center



图 5-10　程序运行结果

在这个程序运行的过程中，用户可以选择四种不同的曲线——“正弦曲线”、“余弦曲线”、“正切曲线”和“余切曲线”。

程序的完整源代码如下所示：

程序清单

```
unit Mutlipage;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  DBCtrls, ComCtrls, Tabnotbk, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    TabbedNotebook1: TTabbedNotebook;
    Image1: TImage;
    Image2: TImage;
    Image3: TImage;
```

Green Apple Data Center

```
    Image4: TImage;
    procedure FormCreate(Sender: TObject);
    procedure Image1Click(Sender: TObject);
    procedure Image2Click(Sender: TObject);
    procedure Image3Click(Sender: TObject);
    procedure Image4Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Image1.Canvas.Pen.Width:=3;
form1.Image2.Canvas.Pen.Width:=3;
form1.Image3.Canvas.Pen.Width:=3;
form1.Image4.Canvas.Pen.Width:=3;
//设置画笔大小
form1.Image1.Transparent:=false;
form1.Image2.Transparent:=false;
form1.Image3.Transparent:=false;
form1.Image4.Transparent:=false;
//设置四个控件具有透明色
end;

procedure TForm1.Image1Click(Sender: TObject);
var
    x1:real;
    y1:real;
    x2:real;
    y2:real;
    n:real;
    //定义变量
begin
x1:=0;
y1:=(form1.image1.Left+form1.Image1.Width)/4;
x2:=x1;
y2:=y1;
form1.Image1.Canvas.MoveTo(trunc(x2),trunc(y2));
```

Green Apple Data Center

```
//定义绘图起始点
n:=0;
while n<3.1415926*6 do
begin
x2:=x2+3.1415926/12;
y2:=y1-40*sin(n);
form1.Image1.Canvas.LineTo(trunc(x2),trunc(y2));
//绘制正弦曲线
n:=n+3.1415926/180;
end;
end;

procedure TForm1.Image2Click(Sender: TObject);
var
    x1:real;
    y1:real;
    x2:real;
    y2:real;
    n:real;
    //定义变量
begin
x1:=0;
y1:=(form1.image2.Left+form1.Image2.Width)/4;
x2:=x1;
y2:=y1;
form1.Image2.Canvas.MoveTo(trunc(x2),trunc(y2));
//绘图起始点
n:=0;
while n<3.1415926*6 do
begin
x2:=x2+3.1415926/12;
y2:=y1-40*cos(n);
form1.Image2.Canvas.LineTo(trunc(x2),trunc(y2));
//绘制余弦曲线
n:=n+3.1415926/180;
end;
end;

procedure TForm1.Image3Click(Sender: TObject);
var x1:real;
    y1:real;
    x2:real;
    y2:real;
    n:real;
    //定义变量
begin
```

Green Apple Data Center

```
x1:=0;
y1:=(form1.image3.Left+form1.Image3.Width)/4;
x2:=x1;
y2:=y1;
form1.Image3.Canvas.MoveTo(trunc(x2),trunc(y2));
//绘图起始点
n:=0;
while n<3.1415926*2 do
begin
x2:=x2+3.1415926/3;
if abs(cos(n))>0.000001 then
begin
y2:=y1-30*sin(n)/cos(n);
form1.Image3.Canvas.LineTo(trunc(x2),trunc(y2));
//绘制正切曲线
end;
n:=n+3.1415926/180;
end;
end;

procedure TForm1.Image4Click(Sender: TObject);
var x1:real;
      y1:real;
      x2:real;
      y2:real;
      n:real;
      //定义变量
begin
x1:=0;
y1:=(form1.image4.Left+form1.Image4.Width)/4;
x2:=x1;
y2:=y1;
form1.Image4.Canvas.MoveTo(trunc(x2),trunc(y2));
//绘图起始点
n:=0;
while n<3.1415926*2 do
begin
x2:=x2+3.1415926/3;
if abs(sin(n))>0.000001 then
begin
y2:=y1-30*cos(n)/sin(n);
form1.Image4.Canvas.LineTo(trunc(x2),trunc(y2));
//绘制余切曲线
end;
n:=n+3.1415926/180;
end;
```

Green Apple Data Center

```
end;
end.
```

5.3 复合窗体开发

如果读者想要设计一个比较复杂的应用程序，那么仅仅利用一个窗体是不够的，这时通常要涉及到复合窗体的程序设计。在复合窗体的程序设计过程中，用户可以对程序中的每一个窗体按照自己的编程意图来设计和编码，并且可以随时的控制每个窗体的显示、隐藏、加载和卸载等操作。

我们已经知道，利用 Delphi 中提供的方法可以绘制出简单的基本图案（如直线、圆和矩形等），同时也可以绘制出比较复杂的图案。下面，我们以一个可以绘制各种图形的复合窗体应用为例来说明复合窗体程序的开发和设计技术。

5.3.1 绘制直线的子窗体设计

下面，我们继续添加绘制直线的子窗体，这个窗体的作用是：当用户在窗体上单击鼠标的左键时，程序就会调用相应的方法在窗体上绘制一条由上一个鼠标单击点到当前鼠标单击点的直线，步骤如下：

1. 新建窗体

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项。在本项目的基础上，选择菜单“ File ”中的“ New Form ”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Standart ”选项后，在“ Button ”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件，添加控件后的窗体如图 5-11 所示。



图 5-11 绘制直线的子窗体

2. 添加代码

在程序的设计过程中，用鼠标的左键双击窗体上的按钮控件，在屏幕上就会弹出一个代码窗口，把光标移动到控件和窗体的相应事件过程中，并且添加如下的程序代码：

程序清单

```
unit Unit1;
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
```

Green Apple Data Center

```
  TForm1 = class(TForm)
    Button1: TButton;
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  start:boolean;
implementation

{$R *.DFM}

procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
if start then
form1.Canvas.LineTo(x,y);
//绘制直线到鼠标的单击点
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
start:=false;
//初始化变量值
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
start:=true;
//设置变量值
end;
end.
```

程序说明：

在程序的变量声明段中，首先定义了一个布尔型变量 Start**，它的作用是控制程序绘制直线的开始时间，在窗体的初始化过程中通过语句** start:=false;**设置为** False**，如果在程序运行的过程中，用户单击按钮“绘制直线”，那么程序就自动的设置变量** Start **的值为** True**，然后通过语句** form1.Canvas.LineTo(x,y);**在窗体上绘制直线。**

5.3.2 绘制椭圆的子窗体设计

下面，我们再创建用于绘制椭圆的子窗体，它的作用是：当用户在子窗体上按下并且移动鼠标时，程序就会调用相应的方法在窗体上绘制一个椭圆。

Green Apple Data Center

具体的程序设计步骤如下：

1．建立窗体

首先启动一个新的项目，选择菜单“ File **”中的“** New Application **”项，在** Delphi **的集成开发环境中就会弹出一个新建的窗体。**



图 5-12 **绘制椭圆子窗体**

2．添加代码

在程序的设计过程用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到到代码段中的适当位置，并且添加如下绘制绘制椭圆的代码：

程序清单

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm2 = class(TForm)
    Label1: TLabel;
    procedure FormCreate(Sender: TObject);
    procedure FormMouseDown(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer);
    procedure FormMouseMove(Sender: TObject; Shift: TShiftState; X,
      Y: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form2: TForm2;
  start:boolean;
```

Green Apple Data Center

```
implementation

{$R *.DFM}

procedure TForm2.FormCreate(Sender: TObject);
begin
start:=false;
//变量初始化
end;

procedure TForm2.FormMouseDown(Sender: TObject; Button: TMouseButton;
   Shift: TShiftState; X, Y: Integer);
begin
start:=true;
//设置变量值
end;

procedure TForm2.FormMouseMove(Sender: TObject; Shift: TShiftState; X,
   Y: Integer);
begin
if start then
    form2.Canvas.Ellipse(x,y,20,20);
    //绘制椭圆
end;

end.
```

以上程序能够实现在程序运行的初始画面中，按下的同时移动鼠标，在窗体上就会绘制出一系列的椭圆，程序的运行结果如图 5-13 所示。

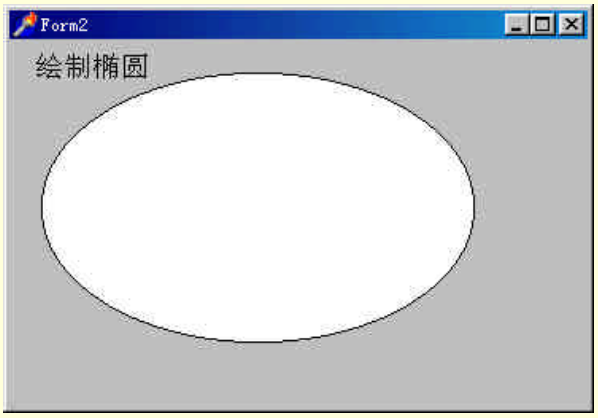


图 5-13　程序运行结果

5.3.3　绘制多边形的子窗体设计

最后，我们再设计第三个用于绘制任意边数的多边形的窗体，它能够实现：在程序运行的过程中，当用户在文本框在中输入边数后单击绘图按钮，程序就会调用相应的方法在窗体上绘制出相应的多边形。

具体的程序设计步骤如下：

Green Apple Data Center

1．新建窗体

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项；

在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Standart ”选项后，在“ Button ”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件，然后接着向窗体上添加一个 Label 控件和一个 Edit 控件，添加控件后的窗体如图 5-14 所示。

其中各个控件的作用如下所示：

- Label 控件：显示一行固定的文本；
- Edit 控件：用于接收用户的输入，从而产生多边形的边数；
- Button 控件：产生绘制多边形的动作。



图 5-14　添加控件后的窗体

2．添加代码

在程序的设计阶段，用鼠标的左键双击窗体，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口中的适当位置，并且添加如下所示的程序代码：

程序清单

```
unit Unit3;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm3 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
```

Green Apple Data Center

```
    public
      { Public declarations }
    end;
var
    Form3: TForm3;

implementation

uses Unit1;


{$R *.DFM}

procedure TForm3.Button1Click(Sender: TObject);
var
      count:integer;
      temp:integer;
      Polyarray:array[0..15] of tpoint;
      //定义变量
begin
      count:=strtoint(form3.edit1.text);
      if count>15 then count:=15;
      for temp:=0 to count do
      begin
      polyarray[temp]:=point(trunc(sin((2*3.14156)*temp/count)*100)+(form3.width div 2),
      trunc(cos((2*3.1415926)*temp/count)*100)+(form3.height div 3));
      end;
      for temp:=count+1 to 15 do
      polyarray[temp]:=polyarray[0];
      form3.Canvas.Polyline(polyarray);
      //绘制多边形
end;

procedure TForm3.FormCreate(Sender: TObject);
begin
form3.Canvas.Pen.Width:=4;
//设置画笔的大小
end;
end.
```

5.3.4 主窗体的完整设计

最后，我们再来建立主窗体，并添加必要的程序代码，以便能由主窗体直接调用上面建立的子窗体程序。

1．新建窗体

首先新建一个窗体，在窗体添加一个 Panel **控件，然后在这个** Panel **控件上再添加三个** Button **控件；之后，继续在窗体上添加第四个按钮控件，添加完控件后的窗体如图** 5-15 **所示。**

Green Apple Data Center

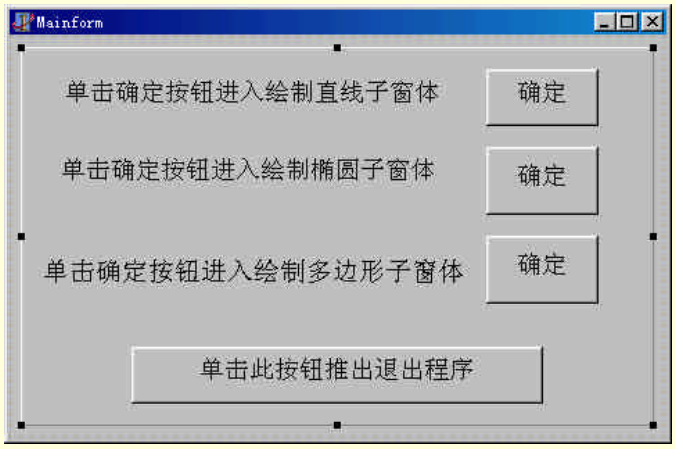


图 5-15　主窗体设计界面

下面是主窗体的属性设置，请读者参考设计自己喜好的主窗体界面：

```
object Mainform: TMainform
  Caption = 'Mainform'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Panel1: TPanel
    Left = 8
    Top = 8
    Width = 457
    Height = 273
    TabOrder = 0
    object Label1: TLabel
      Left = 32
      Top = 24
      Width = 270
      Height = 24
      Caption = '单击确定按钮进入绘制直线子窗体'
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -19
      Font.Name = 'MS Sans Serif'
      Font.Style = []
      ParentFont = False
    end
    object Label2: TLabel
      Left = 24
      Top = 80
```

Green Apple Data Center

```
    Width = 285
    Height = 24
    Caption = '  单击确定按钮进入绘制椭圆子窗体    '
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -19
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
  end
  object Label3: TLabel
    Left = 16
    Top = 152
    Width = 324
    Height = 19
    Caption = '单击确定按钮进入绘制多边形子窗体    '
    Font.Charset = GB2312_CHARSET
    Font.Color = clWindowText
    Font.Height = -19
    Font.Name = '宋体'
    Font.Style = []
    ParentFont = False
  end
  object Button1: TButton
    Left = 336
    Top = 16
    Width = 81
    Height = 41
    Caption = '确定'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -19
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 0
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 336
    Top = 72
    Width = 81
    Height = 49
    Caption = '确定'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
```

Green Apple Data Center

```
      Font.Height = -19
      Font.Name = 'MS Sans Serif'
      Font.Style = []
      ParentFont = False
      TabOrder = 1
      OnClick = Button2Click
    end
    object Button3: TButton
      Left = 336
      Top = 136
      Width = 81
      Height = 49
      Caption = '确定'
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -19
      Font.Name = 'MS Sans Serif'
      Font.Style = []
      ParentFont = False
      TabOrder = 2
      OnClick = Button3Click
    end
  end
  object Button4: TButton
    Left = 88
    Top = 224
    Width = 297
    Height = 41
    Caption = '单击此按钮推出退出程序'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -19
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 1
    OnClick = Button4Click
  end
end
```

2．添加代码

在程序的设计阶段，用鼠标的左键双击窗体，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口中的适当位置。

分别双击进入直线绘制的“确定”按钮、进入圆形绘制的“确定”按钮、进入多边形绘制的“确定”按钮和“退出”按钮，添加如下所示的程序代码：

```
procedure TMainform.Button1Click(Sender: TObject);
begin
```

Green Apple Data Center

```
form1.show();     //启动调用并显示 form1 子窗体
end;

procedure TMainform.Button2Click(Sender: TObject);
begin
form2.show();     //启动调用并显示 form3 子窗体
end;
procedure TMainform.Button3Click(Sender: TObject);
begin
form3.show();     //启动调用并显示 form3 子窗体
end;

procedure TMainform.Button4Click(Sender: TObject);
begin
close();
end;
```

3．运行程序

代码添加完毕，我们将本次创建的应用程序保存下来（这里，项目保存为 P_Mutiform**，主窗体的单元文件保存为** U_Multiform**，绘制直线子窗体的单元文件保存为** unit1**，绘制椭圆子窗体的单元文件保存为** unit2**，绘制多边形子窗体的单元文件保存为** unit3**）。**

按 F9 **键运行程序，如图** 5-16 **左所示为初始主窗体画面。**

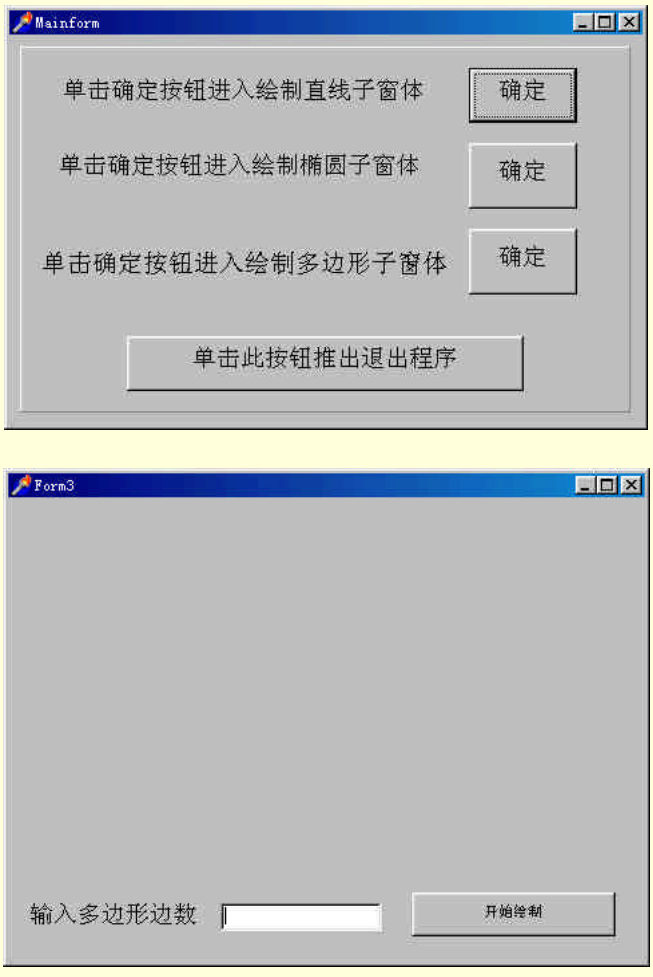


图 5-16　**程序运行主窗体和子窗体**

Green Apple Data Center

单击进入绘制多边形窗体的“确定”按钮，就会打开绘制多边形子窗体，如图 5-11 右所示。在本子窗体上，输入参数为 5，就会得到程序自动绘制的图形，程序的运行结果如图 5-17 所示。

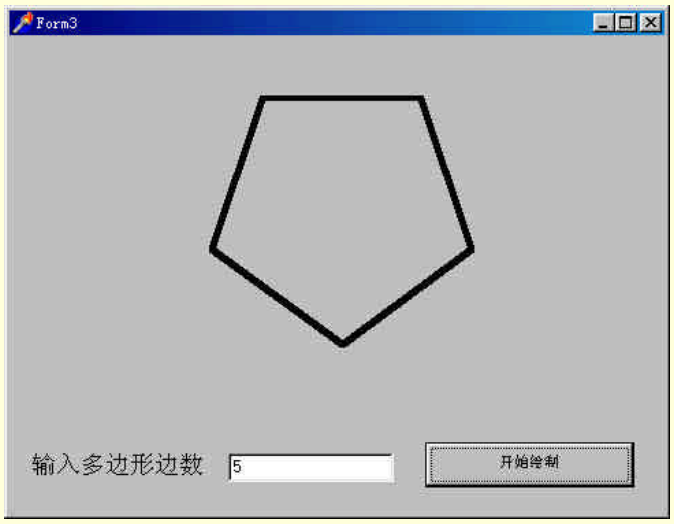


图 5-17　程序运行结果

5.4　多文档界面窗体的开发

什么是多文档界面窗体呢？其实，基本上所有的用户都使用过，比如 Microsoft Word　97/2000。

那么又怎样来判断一个 Windows 应用程序是否拥有 MDI 用户界面呢？一个简单的方法就是判断应用程序是否包含有能在窗口或文档之间进行切换的子菜单的 Windows 菜单项，如果有，那么该应用程序就是 MDI 应用程序，以 MDI 作为用户界面，否则就不是 MDI 程序。

下面我们就通过一个简单的示例程序向读者说明，在 Delphi 中如何设计一个 MDI 应用程序，为了说明 SDI 与 MDI 之间的区别，我们仍然以字处理程序为例来说明 MDI 程序设计的常用方法和一般技巧，具体的程序设计步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Standard 选项后，向窗体上添加一个 ActionList 控件和一个 MainMenu 控件，然后在控件工具栏上选择 Win32 选项，并且向窗体上添加一个 ToolBar 控件、一个 StatusBar 控件和一个 ImageList 控件，在控件工具栏上选择 Additional 选项，向窗体上添加十一个 SpeedButton 控件，最后选择控件工具栏上的 Dialogs 选项，向窗体上添加一个 OpenDialog 控件。

添加控件后的窗体如图 5-18 所示，各个控件的功能将在后面分别的加以介绍。

Green Apple Data Center

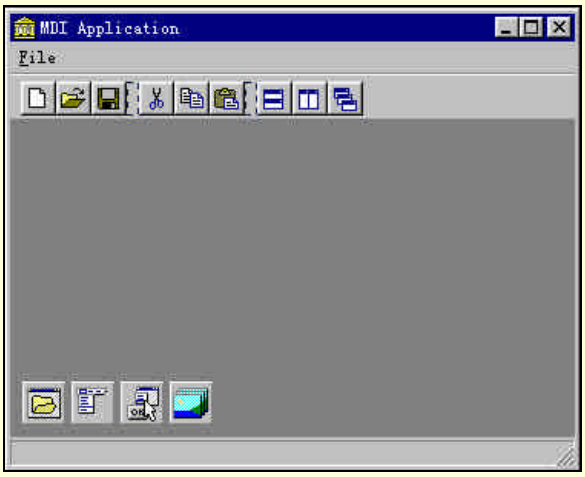


图 5-18　添加控件后的窗体

添加到窗体上各个控件的作用如下所示：

- StatusBar 控件：在程序运行的过程中显示应用程序当前的状态和各个控件的提示文本；
- MainMenu 控件：为程序运行提供菜单设计的容器；
- ActionList 控件：为程序运行提供一些标准和非标准操作；
- ImageList 控件：在其中装有一些图像文件，可以显示显示在菜单和工具栏中；
- OpenDialog 控件：显示一个对话框，在其中可以选择一个待打开的文本文件；
- 工具栏控件：在其中可以设计菜单的快捷操作方式。

2. 菜单设计

在程序设计的过程中用鼠标双击窗体上的 MainMenu 控件，在窗体上就会弹出一个菜单编辑窗口，向当前的窗体上添加一个主菜单项 File，并且在 File 菜单项下添加 New、Open 和 Exit 三个子菜单项，添加到窗体上 File 菜单中各个子菜单项的作用如下所示：

- New 菜单：创建一个新的文件；
- Open 菜单：显示一个打开文件的对话框，并且可以根据用户的选择打开一个文本文件；
- Exit 菜单：结束程序的运行。

3. ActionList 控件

添加到窗体上的 ActionList 控件的作用是在程序中为设计人员提供一些标准的操作，如 TWindowClose、TWindowCascade 和 TWindowArrange 等。在程序设计的过程中，用户可以随时的向当前的应用程序中添加一些标准操作。

在本示例程序的设计过程中，用鼠标双击窗体上的 ActionList 控件，在窗体上就会弹出一个如图 5-19 所示的对话框。

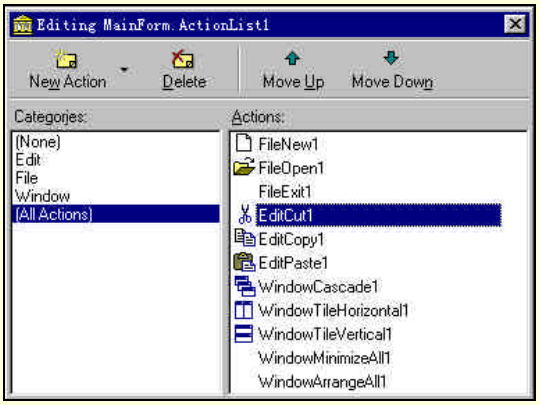


图 5-19　添加标准操作

单击工具栏上 New Action 选项右侧的下三角，就会弹出一个 New Standard Action 选项，单击这个选项，在窗体上就会弹出一个显示所有标准操作的对话框，在其中添加 EditCut、EditCopy、EditPaste、WindowCascade、

Green Apple Data Center

WindowTileHorizontal、WindowTileVertical、WindowMinimizeAll 和 WindowArrangeAll 等八个标准的操作，并且添加 FileNew、FileOpen、FileExit 等三个用户自定义的操作。

EditCut、EditCopy、EditPaste、WindowCascade、WindowTileHorizontal、WindowTileVertical、WindowMinimizeAll 和 WindowArrangeAll 等八个标准的操作的响应代码已经封装在 ActionList 控件中，不需要用户自己编写，但是 FileNew、FileOpen 和 FileExit 等三个用户自定义的操作的响应代码却需要用户自己添加。

为 FileNew 菜单项添加代码如下所示：

```
procedure TMainForm.FileNew1Execute(Sender: TObject);
begin
  CreateMDIChild('NONAME' + IntToStr(MDIChildCount + 1));
  //新建一个空白窗口
end;
```

为 FileOpen 菜单项添加代码如下所示：

```
procedure TMainForm.FileOpen1Execute(Sender: TObject);
begin
  if OpenDialog.Execute then
     //显示对话框
    CreateMDIChild(OpenDialog.FileName);
    //打开指定的文件
end;
```

为 FileExit 菜单项添加代码如下所示：

```
procedure TMainForm.FileExit1Execute(Sender: TObject);
begin
  Close;
  //结束程序的运行
end;
```

4. 响应主窗体事件

在程序设计的过程中，用鼠标左键双击窗体上的空白处，在弹出的代码窗口中选择窗体的 FormCreate 事件，并且添加如下所示的响应代码：

程序清单（主窗体程序代码）

```
unit MDI;
interface
uses Windows, SysUtils, Classes, Graphics, Forms, Controls, Menus,
  StdCtrls, Dialogs, Buttons, Messages, ExtCtrls, ComCtrls, StdActns,
  ActnList, ToolWin, ImgList;
type
  TMainForm = class(TForm)
    N1: TMenuItem;
    FileExitItem: TMenuItem;
    OpenDialog: TOpenDialog;
    StatusBar: TStatusBar;
    ActionList1: TActionList;
    EditCut1: TEditCut;
    EditCopy1: TEditCopy;
    EditPaste1: TEditPaste;
```

Green Apple Data Center

```
    FileNew1: TAction;
    FileSave1: TAction;
    FileExit1: TAction;
    FileOpen1: TAction;
    FileSaveAs1: TAction;
    WindowCascade1: TWindowCascade;
    WindowTileHorizontal1: TWindowTileHorizontal;
    WindowArrangeAll1: TWindowArrange;
    WindowMinimizeAll1: TWindowMinimizeAll;
    HelpAbout1: TAction;
    FileClose1: TWindowClose;
    WindowTileVertical1: TWindowTileVertical;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    FileNewItem: TMenuItem;
    FileOpenItem: TMenuItem;
    ToolBar2: TToolBar;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    ToolButton3: TToolButton;
    ToolButton4: TToolButton;
    ToolButton5: TToolButton;
    ToolButton6: TToolButton;
    ToolButton9: TToolButton;
    ToolButton7: TToolButton;
    ToolButton8: TToolButton;
    ToolButton10: TToolButton;
    ToolButton11: TToolButton;
    ImageList1: TImageList;
    procedure FileNew1Execute(Sender: TObject);
    procedure FileOpen1Execute(Sender: TObject);
    procedure FileExit1Execute(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
    procedure CreateMDIChild(const Name: string);
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;
implementation
{$R *.DFM}
uses ChildWin;
```

Green Apple Data Center

```
procedure TMainForm.CreateMDIChild(const Name: string);
var
  Child: TMDIChild;
  //变量声明
begin
  Child := TMDIChild.Create(Application);
  //创建窗口
  Child.Caption := Name;
  if FileExists(Name) then Child.Memo1.Lines.LoadFromFile(Name);
  //打开指定文件
end;

procedure TMainForm.FileNew1Execute(Sender: TObject);
begin
  CreateMDIChild('NONAME' + IntToStr(MDIChildCount + 1));
  //新建一个空白窗口
end;

procedure TMainForm.FileOpen1Execute(Sender: TObject);
begin
  if OpenDialog.Execute then
     //显示对话框
    CreateMDIChild(OpenDialog.FileName);
    //打开指定的文件
end;

procedure TMainForm.FileExit1Execute(Sender: TObject);
begin
  Close;
  //结束程序的运行
end;

procedure TMainForm.FormCreate(Sender: TObject);
begin
OpenDialog.Title:='请选择一个文本文件:';
//设置对话框标题
OpenDialog.InitialDir:='c:\pwin98';
//设置对话框缺省标题
OpenDialog.Filter:='All files(*.*)|*.*|Text files(*.txt)|*.txt';
//设置文件过滤器
end;
end.
```

5. 添加子窗体

既然是多文档程序设计，当然就要涉及到子窗体的设计了。在在程序设计的过程中，用鼠标左键选择菜单 File 中的 New Form 选项，在当前的项目中就会自动的添加一个新的空白窗体，向新增窗体上添加一个 MainMenu

Green Apple Data Center

控件、一个 SaveDialog 控件和一个 Memo 控件，并且同时进行子窗体的菜单设计。添加控件后的子窗体如图 5-20 所示。

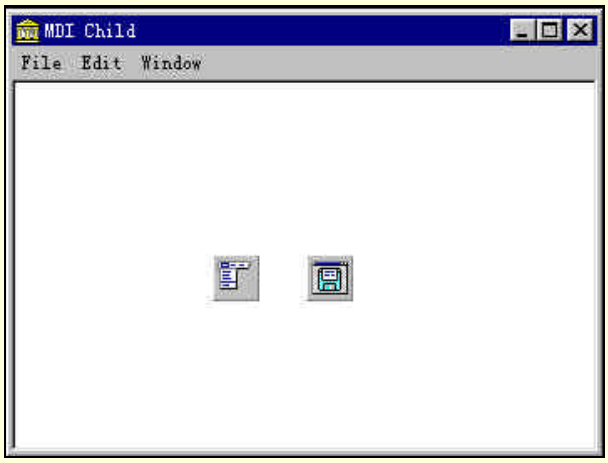


图 5-20　添加控件后的子窗体

注意：

✍　在程序运行的过程中，如果激活了子窗体，那么子窗体中的菜单将会自动的覆盖父窗体上的菜单。

子窗体上各个控件的属性设置如下所示：

```
object MDIChild: TMDIChild
  Caption = 'MDI Child'
  FormStyle = fsMDIChild
  Menu = MainMenu1
  OnClose = FormClose
  OnCreate = FormCreate
  object Memo1: TMemo
    Left = 0
    Top = 0
    Width = 282
    Height = 161
    Align = alClient
  end
  object MainMenu1: TMainMenu
    Images = MainForm.ImageList1
    Left = 128
    Top = 80
    object File1: TMenuItem
      Caption = 'File'
      object New1: TMenuItem
        Action = MainForm.FileNew1
      end
      object Open1: TMenuItem
        Action = MainForm.FileOpen1
      end
      object Close1: TMenuItem
        Caption = 'Close'
        OnClick = Close1Click
```

Green Apple Data Center

```
      end
      object Save1: TMenuItem
        Caption = 'Save'
        ImageIndex = 5
        OnClick = Save1Click
      end
    end
    object Edit1: TMenuItem
      Caption = 'Edit'
      object Cut1: TMenuItem
        Action = MainForm.EditCut1
      end
      object Copy1: TMenuItem
        Action = MainForm.EditCopy1
      end
      object Paste1: TMenuItem
        Action = MainForm.EditPaste1
      end
    end
    object Window1: TMenuItem
      Caption = 'Window'
      object Cascade1: TMenuItem
        Action =
        MainForm.WindowCascade1
      end
      object TileHorizontal1: TMenuItem
        Action =
        MainForm.WindowTileHorizontal1
      end
      object TileVertical1: TMenuItem
        Action =
        MainForm.WindowTileVertical1
      end
      object ArrangeAll11: TMenuItem
        Action =
        MainForm.WindowArrangeAll1
      end
      object MinimizeAll1: TMenuItem
        Action =
        MainForm.WindowMinimizeAll1
      end
    end
  end
  object SaveDialog1: TSaveDialog
    Left = 184
    Top = 80
```

Green Apple Data Center

```
  end
end
```

在子窗体上有三个菜单项——File、Edit 和 Window，其中添加到窗体上的 File 菜单中各个子菜单项的作用如下所示：

- New **菜单：创建一个新的文档窗口；**
- Open **菜单：显示一个打开文件的对话框，用户可以有选择的打开一个文件，并且在父窗体中会打开一个新的子窗体来显示选中文件的内容；**
- Close **菜单：关闭当前打开的文档窗口，如果用户对文档做出了修改，那么程序将会提示用户是否保存；**
- Save **菜单：显示一个存储文件的对话框，并且可以根据用户的选择保存当前打开或编辑过的文档。**
- **在 Edit 主菜单下包含有三个子菜单，它们是 Cut、Copy 和 Paste，它们的作用如下所示：**
- Cut **菜单：将当前选中的内容剪切到剪贴板上；**
- Copy **菜单：将当前选中的内容复制到剪贴板上；**
- Paste **菜单：将当前剪贴板上的内容放置到当前位置上。**

在 Window 主菜单下包含有五个子菜单项，它们是 Cascade、Tile Horizontally、Tile Vertically、Arrange All 和 Minimize All。

在程序运行的过程中，当打开一个以上的文档窗口之后，就涉及到各个文档窗口之间的排列问题，用户可以通过用鼠标单击菜单 Windows 中的各个子菜单项来选择文档窗口的排列方式，在本程序中可以选择的文档窗口排列方式有层叠显示、横向平铺显示、纵向平铺显示、排列图标和所有文档最小化等。

按照下面 MDI 程序设计中的程序清单为子窗体添加代码：

程序清单（子窗体的响应代码）

```
unit MDIChild;

interface

uses Windows, Classes, Graphics, Forms, Controls, StdCtrls, Menus, Dialogs;

type
  TMDIChild = class(TForm)
    Memo1: TMemo;
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    New1: TMenuItem;
    Open1: TMenuItem;
    Close1: TMenuItem;
    Save1: TMenuItem;
    Edit1: TMenuItem;
    Cut1: TMenuItem;
    Copy1: TMenuItem;
    Paste1: TMenuItem;
    Window1: TMenuItem;
    Cascade1: TMenuItem;
    TileHorizontal1: TMenuItem;
    TileVertical1: TMenuItem;
    ArrangeAll11: TMenuItem;
```

Green Apple Data Center

```
    MinimizeAll1: TMenuItem;
    SaveDialog1: TSaveDialog;
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Close1Click(Sender: TObject);
    procedure Save1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

implementation

uses Main;
{$R *.DFM}

procedure TMDIChild.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  if memo1.Modified then
     //如果文档被修改
     if SaveDialog1.Execute then
        //显示对话框
        memo1.Lines.SaveToFile(SaveDialog1.FileName);
        //存储文件
  Action := caFree;
  //关闭窗口
end;

procedure TMDIChild.Close1Click(Sender: TObject);
begin
main.MainForm.ActiveMDIChild.Close;
//关闭当前活动的窗口
end;

procedure TMDIChild.Save1Click(Sender: TObject);
begin
if SaveDialog1.Execute then
   memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;

procedure TMDIChild.FormCreate(Sender: TObject);
begin
SaveDialog1.Title:='请选择一个文本文件:';
//设置对话框标题
SaveDialog1.InitialDir:='c:\pwin98';
```

Green Apple Data Center

```
//设置对话框缺省标题
SaveDialog1.Filter:='All files(*.*)|*.*|Text files(*.txt)|*.txt';
//设置文件过滤器
end;


end.
```

6. 运行程序

按照以上附加的程序清单添加剩余的程序代码。

做完以上的工作后存储文件，按键盘上的功能键 F9 运行程序。

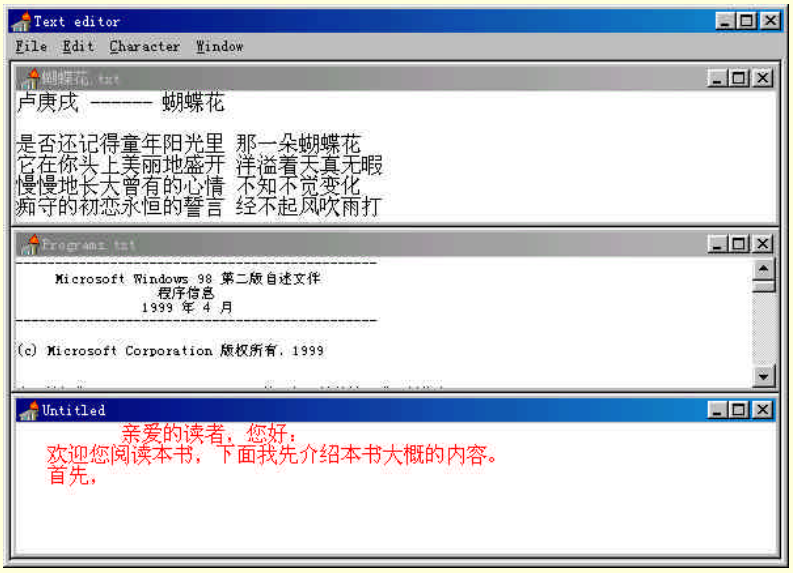


图 5-21 程序运行结果

在程序运行的过程中，用户即可以通过菜单来进行各种文件操作，如新建一个文本文件、打开文本文件和存储文件等，也可以进行剪切、复制和粘贴等编辑操作，对打开的多个文档窗口进行排列，当然还可以通过工具栏上快捷按钮直接进行操作。程序运行结果如图 5-21 所示。

5.5 小 结

通过本章的学习，用户可以掌握用户窗体设计的一般方法和常用技巧，如文档窗体设计、多页窗体设计和复合窗体设计等。

在介绍复合窗体设计技术的同时，还进一步向读者介绍图形的绘制技术，读者可以把本章的内容当作对上一章内容的复习，并在练习的基础上学习新的窗体开发技术。希望读者能够熟练的利用本章介绍的知识，自己独立设计出丰富多彩、功能多样的用户窗体应用程序。

Green Apple Data Center

第六章　常见 VCL 控件的使用

在 Delphi 中包括很多的控件，同时又有几个最为基本、也是最常用的可视化控件（VCL 控件）：Button 控件、Bitbtn 控件、Edit 控件、Label 控件、Memo 控件、CheckBox 控件和 RadioButton 控件。

熟练的运用这些控件，是设计更加复杂多用的应用程序的基础。在本章中，我们将以几个示例程序的形式来向读者说明各个控件的使用方法。

6.1　Edit 控件和 Button 控件

在 Delphi 中的按钮控件不但可以充当事件响应的容器，同时还可以在其中实现显示位图等功能，在本节中通过一个实例程序——“ 简易计算器 ”来向读者介绍形式各异的按钮控件的用法。下面就利用 Button 控件和 Edit 控件来制作一个“ 计算器 ”应用程序，它能够实现整数的简单的加减乘除运算。具体的程序设计步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项；

然后，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Standart ”选项后，在“ Button ”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件，接着向窗体上添加十六个 Button 控件和一个 Edit 控件，各个控件的功能将在后面分别的加以介绍。添加控件后的窗体如图 6-1 所示。



图 6-1　添加控件后的窗体

在上面所添加的 16 个 Button 控件中，可以大致的分为三类——字符类、运算类和清零类，下面分别的加以介绍：

（1）字符类 Button 控件

所谓的字符类 Button 控件，即 10 个 Caption 属性设置为从 0—9 的 Button 控件，它们的作用是接收用户在程序运行过程中的输入，它们的属性设置如下所示：

```
object Button1: TButton
  Left = 40
  Top = 80
  Width = 43
  Height = 25
  Caption = '1'
```

Green Apple Data Center

```
    TabOrder = 1
end
object Button2: TButton
    Left = 112
    Top = 80
    Width = 43
    Height = 25
    Caption = '2'
    TabOrder = 2
end
object Button3: TButton
    Left = 192
    Top = 80
    Width = 43
    Height = 25
    Caption = '3'
    TabOrder = 3
end
object Button4: TButton
    Left = 40
    Top = 136
    Width = 43
    Height = 25
    Caption = '4'
    TabOrder = 4
end
object Button5: TButton
    Left = 112
    Top = 136
    Width = 43
    Height = 25
    Caption = '5'
    TabOrder = 5
end
object Button6: TButton
    Left = 192
    Top = 136
    Width = 43
    Height = 25
    Caption = '6'
    TabOrder = 6
end
object Button7: TButton
    Left = 40
    Top = 192
    Width = 43
```

Green Apple Data Center

```
    Height = 25
    Caption = '7'
    TabOrder = 7
  end
  object Button8: TButton
    Left = 112
    Top = 192
    Width = 43
    Height = 25
    Caption = '8'
    TabOrder = 8
  end
  object Button9: TButton
    Left = 192
    Top = 192
    Width = 43
    Height = 25
    Caption = '9'
    TabOrder = 9
  end
  object Button10: TButton
    Left = 40
    Top = 248
    Width = 43
    Height = 25
    Caption = '0'
    TabOrder = 10
  End
```

其实在它们的属性设置中，还应该包括如 Left、Top、Width **和** Height **等属性的设置，但是由于篇幅的原因，在这里就不多加介绍了，读者可以从添加控件后的窗体中看出它们之间的相对位置关系。**

（2）运算类 Button **控件**

所谓的运算类 Button **控件指的是五个分别能够完成加、减、乘、除、输出运算结果的** Button **控件，它们的属性设置如下所示：**

```
object Button11: TButton
    Left = 112
    Top = 248
    Width = 43
    Height = 25
    Caption = '='
    TabOrder = 11
  end
  object Button13: TButton
    Left = 272
    Top = 88
    Width = 75
    Height = 25
```

Green Apple Data Center

```
    Caption = '+'
    TabOrder = 13
  end
  object Button14: TButton
    Left = 272
    Top = 136
    Width = 75
    Height = 25
    Caption = '—'
    TabOrder = 14
  end
  object Button15: TButton
    Left = 272
    Top = 192
    Width = 75
    Height = 25
    Caption = 'X'
    TabOrder = 15
  end
  object Button16: TButton
    Left = 272
    Top = 240
    Width = 75
    Height = 25
    Caption = '/'
    TabOrder = 16
  End
```

与列举字符类 Button 控件相同，操作类 Button 控件的属性也只列举了 Caption 属性，如果读者有什么不清楚的地方，请参看添加控件后的窗体。

（3）清零类 Button 控件

这里所谓的清零按钮就是指运算的复位（Reset 按钮），目的是让文本框中显示的文本清空。它的属性设置如下所示：

```
  object Button12: TButton
    Left = 192
    Top = 248
    Width = 43
    Height = 25
    Caption = 'Reset'
    TabOrder = 12
  end
```

2．程序的初始化

由于在程序运行的过程中，要进行诸如存储数据等操作，所以在窗体的声明段中要声明三个变量 temp1、temp2 和 flag。

```
var
  Form1: TForm1;
  temp1: string;
```

Green Apple Data Center

```
  temp2: real;
  flag: integer;
```

其中变量 temp1、temp2 用做中间变量，即临时存储数据，而整型变量 flag 则用于在程序运行的过程中判断用户选择的运算类型，即用户到底选择的是加法运算还是减法运算等。

在程序的初始化过程中，还应该包括窗体 FormCreate()事件的初始化，在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Edit1.Clear;
//清空文本框
end;
```

窗体 FormCreate()事件中的语句 form1.Edit1.Clear;用来在程序运行的初期清空文本框，以接收用户的输入。

3．响应数字输入

在程序运行的过程中，用户在计算器上的键盘（字符类 Button 控件）上按下鼠标的左键时，在文本框中就应该能够随时的显示出用户的输入，为了实现这一功能，需要在十个字符类 Button 控件的事件响应中添加如下代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'1'
//单击 1
end;
……

procedure TForm1.Button10Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'0'
//单击 0
end;
```

由于篇幅的关系，在这里仅仅列举了字符类控件 Button1 和 Button10 的事件响应代码，其它几个控件的响应代码请读者参见附后的源程序代码。

4．响应运算符输入

在本示例程序中，共有四种运算符——“ +、-、*、/ ”，在程序运行的过程中，单击这四个运算符，就可以实现相应的加减乘除运算，下面以加法运算为例来说明在程序中如何实现简单的数学运算。

```
procedure TForm1.Button13Click(Sender: TObject);
begin
flag:=1;
//设置标志变量值
temp1:=edit1.Text;
//为变量 temp1 赋值
edit1.Clear
//清空文本框
end;
```

如果用户在程序运行的过程中，用鼠标的左键单击“ + ”，程序首先会设置标志变量 flag 的值为 1，也就表示程序将要进行的数学运算为加法运算，然后通过语句 temp1:=edit1.Text;把当前文本框中的数据存储在临时变量 temp1 中，最后清空文本框。

需要注意的是，其它几个运算符的代码实现过程请读者参看附后的源程序。

Green Apple Data Center

5. 输出运算结果

最后程序运算的结果要输出到文本框中，但是其中的一个难点就在于如何判断用户所选择的算法类型，在本程序中，标志变量 flag 中存储着用户的选择类型，所以在“ = ”按钮的响应代码中通过一个选择语句来得到程序的运算结果：

```
procedure TForm1.Button11Click(Sender: TObject);
begin
    case flag of
    1: temp2:=strtofloat(edit1.text)+strtofloat(temp1);
    2: temp2:=strtofloat(temp1)-strtofloat(edit1.text);
    3: temp2:=strtofloat(edit1.text)*strtofloat(temp1);
    4: temp2:=strtofloat(temp1)/strtofloat(edit1.text);
    end;
edit1.Text:=floattostr(temp2) ;
end;
```

值得说明，临时变量 temp2 中存储的程序运算结果为 Float 类型，而文本框的输出类型为 String 类型，所以在程序中要调用一个 floattostr()函数进行数据类型的转换，最后通过语句 edit1.Text:=floattostr(temp2) ;来输出程序的运算结果。

6. 运行程序

按照附后的源程序，添加剩余的程序代码后，选择菜单“ File ”中的“ Save All ”选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序运行的初始画面如图 6-2 所示。



图 6-2 程序运行的初始画面

在程序运行的过程中，用鼠标单击窗体上的数字按钮，输入数据 1234，然后单击运算符“ × ”，再次输入数据 5678，单击“ = ”，得到程序运算运行结果为 7006652，如图 6-3 所示。

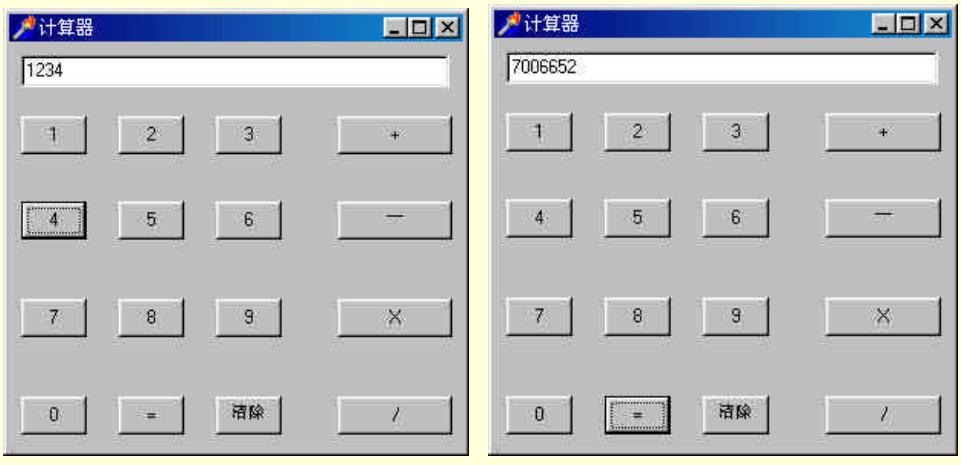


图 6-3 程序运算的结果

Green Apple Data Center

在这个示例应用程序中，不但可以实现加法运算，还可以实现减法、乘法和除法的运算，程序完整的源代码如下所示：

程序清单

```
unit calculator;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button8: TButton;
    Button9: TButton;
    Button10: TButton;
    Button11: TButton;
    Button12: TButton;
    Button13: TButton;
    Button14: TButton;
    Button15: TButton;
    Button16: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button10Click(Sender: TObject);
    procedure Button11Click(Sender: TObject);
    procedure Button13Click(Sender: TObject);
    procedure Button14Click(Sender: TObject);
    procedure Button15Click(Sender: TObject);
```

Green Apple Data Center

```
    procedure Button16Click(Sender: TObject);
    procedure Button12Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  temp1: string;
  temp2: real;
  flag: integer;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Edit1.Clear;
//清空文本框
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'1'
//单击 1
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'2'
//单击 2
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'3'
//单击 3
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'4'
//单击 4
```

Green Apple Data Center

```
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'5'
//单击 5
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'6'
//单击 6
end;

procedure TForm1.Button7Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'7'
//单击 7
end;

procedure TForm1.Button8Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'8'
//单击 8
end;

procedure TForm1.Button9Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'9'
//单击 9
end;

procedure TForm1.Button10Click(Sender: TObject);
begin
edit1.Text:=edit1.Text+'0'
//单击 0
end;

procedure TForm1.Button11Click(Sender: TObject);
begin
case flag of
        1: temp2:=strtofloat(edit1.text)+strtofloat(temp1);
        2: temp2:=strtofloat(temp1)-strtofloat(edit1.text);
        3: temp2:=strtofloat(edit1.text)*strtofloat(temp1);
        4: temp2:=strtofloat(temp1)/strtofloat(edit1.text);
end;
```

Green Apple Data Center

```
edit1.Text:=floattostr(temp2) ;
end;

procedure TForm1.Button13Click(Sender: TObject);
begin
flag:=1;
//设置标志变量值
temp1:=edit1.Text;
//为变量 temp1 赋值
edit1.Clear
//清空文本框
end;

procedure TForm1.Button14Click(Sender: TObject);
begin
flag:=2;
//设置标志变量值
temp1:=edit1.Text;
//为变量 temp1 赋值
edit1.Clear
//清空文本框
end;

procedure TForm1.Button15Click(Sender: TObject);
begin
flag:=3;
//设置标志变量值
temp1:=edit1.Text;
//为变量 temp1 赋值
edit1.Clear
//清空文本框
end;

procedure TForm1.Button16Click(Sender: TObject);
begin
flag:=4;
//设置标志变量值
temp1:=edit1.Text;
//为变量 temp1 赋值
edit1.Clear
//清空文本框
end;

procedure TForm1.Button12Click(Sender: TObject);
begin
edit1.Clear
```

Green Apple Data Center

```
//清除文本框中的文字
end;
end.
```

6.2 Listbox 控件和 Combobox 控件

在进行文本处理的过程中，我们经常要与字体打交道，在这里所讲的字体不是指字的大小，也不是指字体的颜色，而是指“宋体”、“隶书”等字体形式，常见的字体选择对话框如图 6-4 所示。



图 6-4 字体选择对话框

在本节中，我们就利用 ListBox 控件和 ComboBox 控件来制作一个应用程序，在其中可以为文本框中的文字选择一个合适的字体，同时可以通过键盘上的字符来选择合适的字体，具体的程序制作步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Standard 选项后，在 Memo 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 控件，接着向窗体上添加一个 ComboBox 控件和一个 ListBox 控件，各个控件的功能在后面再加以介绍，各个控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 391
  Height = 269
  Caption = 'Form1'
  Color = clBtnFace
  OnCreate = FormCreate
  object ComboBox1: TComboBox
    Left = 16
    Top = 16
    Width = 169
    Height = 24
    Text = '系统字体:'
    OnClick = ComboBox1Click
  end
  object ListBox1: TListBox
    Left = 16
```

Green Apple Data Center

```
    Top = 48
    Width = 169
    Height = 177
    OnClick = ListBox1Click
    OnDrawItem = ListBox1DrawItem
    OnMeasureItem =
    ListBox1MeasureItem
  end
  object Memo1: TMemo
    Left = 192
    Top = 16
    Width = 169
    Height = 209
  end
end
```

添加控件后的窗体如图 6-5 **所示。**



图 6-5　**添加控件后的窗体**

窗体上添加的控件作用如下所示：

- ListBox1 **控件：显示系统的字体；**
- ComboBox1 **控件：显示系统的字体；**
- Memo1 **控件：显示文本文件。**

2．**程序的初始化**

程序的初始化过程，也就是对窗体 FormCreate()**事件的初始化。在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到** FormCreate()**事件的过程处理代码中，并且添加如下所示代码：**

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  form1.Listbox1.Items := Screen.Fonts;
  //Listbox 控件的初始化
  form1.ComboBox1.Items:=screen.Fonts;
  //ComboBox 控件的初始化
  form1.Memo1.Lines.LoadFromFile('c:\pwin98\win.ini');
  //读取文件信息
end;
```

在程序运行的初期，首先激活窗体 FormCreate()**事件中的代码，执行语句**

Green Apple Data Center

Listbox1.Items := Screen.Fonts;，ComboBox1.Items:=screen.Fonts;

在控件 Listbox1 **和** ComboBox1 **中显示系统当前所有字体，然后通过语句**

form1.Memo1.Lines.LoadFromFile('c:\pwin98\win.ini');

在文本控件中显示文件 c:\pwin98\win.ini **的内容。**

其中文本文件 c:\pwin98\win.ini **为** Windows **操作系统的启动文件，用户可以在其中设置各项参数，文件的内容如下所示：**

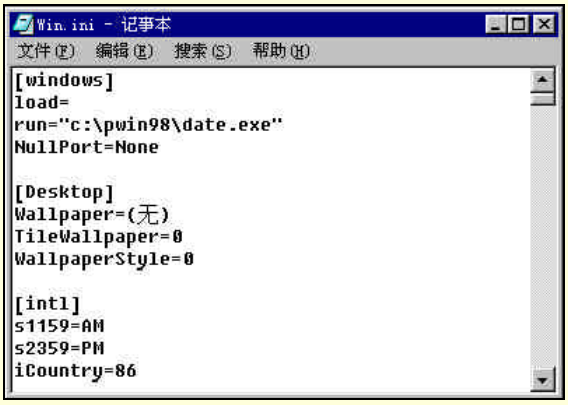


图 6-6 Win.ini **的文件内容**

3. **添加字体选择代码**

在程序运行的过程中，当用户在 ComboBox **控件中选择一个字体时，在** ListBox **控件中应该显示出当前选中的字体，同时在** Memo **控件中可以应用当前选中的字体显示指定的文本文件，为了实现以上的功能，在程序设计过程中添加下列代码：**

```
procedure TForm1.ListBox1Click(Sender: TObject);
begin
  memo1.Font.Name:=ListBox1.Items[ListBox1.ItemIndex];
  //为 Memo 控件应用选中的字体
  comboBox1.ItemIndex:=listbox1.itemindex;
  //显示相应的关联选项
end;

procedure TForm1.ComboBox1Click(Sender: TObject);
begin
 ListBox1.Items[ListBox1.ItemIndex]:=combobox1.Items[combobox1.itemindex];
  //显示相应的关联选项
  memo1.Font.Name:=ListBox1.Items[ListBox1.ItemIndex];
  //为 Memo 控件应用选中的字体
end;
```

4. **添加键盘响应代码**

用户不但可以在程序运行的过程中用鼠标来选择字体，还可以在键盘上按首字母来选择相应的字体，为此在程序中还要添加能够对键盘作出响应的代码。

在 ListBox **控件的** ListBox1MeasureItem ()**事件和窗体的** DrawItem()**事件中添加如下所示的响应键盘动作的代码：**

```
procedure TForm1.ListBox1DrawItem(Control: TWinControl; Index: Integer;
  Rect: TRect; State: TOwnerDrawState);
begin
  with ListBox1.Canvas do
```

Green Apple Data Center

```
  begin
    FillRect(Rect);
    Font.Name := ListBox1.Items[Index];
    Font.Size := 0;
    TextOut(Rect.Left+1, Rect.Top+1, ListBox1.Items[Index]);
  end;
end;

procedure TForm1.ListBox1MeasureItem(Control: TWinControl; Index: Integer;
  var Height: Integer);
begin
  with ListBox1.Canvas do
  begin
    Font.Name := Listbox1.Items[Index];
    Font.Size := 0;
    Height := TextHeight('Wg') + 2;
  end;
end;
```

5. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，在程序运行的过程中，通过键盘或者鼠标在列表框中选择一个字体，这时的文本框中的文字将会以选定的字体显示，运行结果如图 6-7 所示。

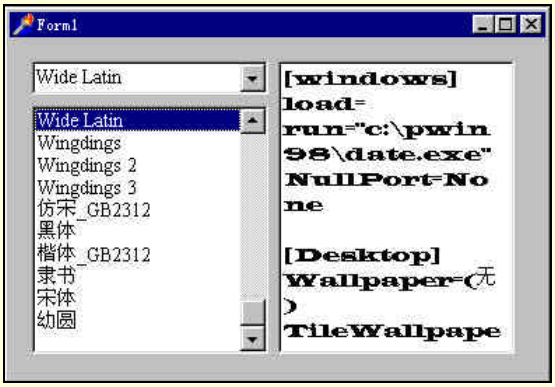


图 6-7 程序运行结果

6.3 Memo 控件和 Dialog 控件

在 Delphi 中，Memo（备忘录）控件的主要作用是显示指定的文本，在使用的过程中，它可以用做文本显示的容器，用于接收用户的输入，同时还可以显示指定的文本文件，所以在一定的程度上，Memo 控件还可以用做文本文件的处理。

下面我们就以 Memo 控件结合对话框 Dialog 控件为例，来说明在 Delphi 中如何进行文本文件的简单处理操作。这个示例程序所能够实现的功能如下所示：

- **可以打开一个典型的文本文件，用户可以在程序运行的过程中选择文件的路径和文件名；**
- **进行简单的文本编辑后，可以把修改结果保存到一个文本文件中；**
- **可以为 Memo 控件中的显示文本选择合适的字体；**

Green Apple Data Center

■ 可以为 Memo 控件选择合适的背景色。

为了能够实现以上的功能，我们设计的程序具体制作步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项；

在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Standart ”选项后，在“ Memo ”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 控件，接着向窗体上添加控件：五个 Button 控件、一个 OpenDialog 控件、一个 SaveDialog 控件、一个 FontDialog 控件和一个 ColorDialog。

添加控件后的窗体如图 6-8 所示。

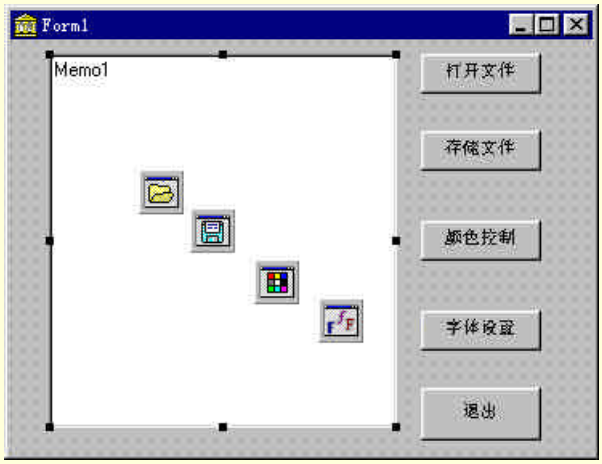


图 6-8 添加控件后的窗体

窗体中各个控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 371
  Height = 284
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Memo1: TMemo
    Left = 24
    Top = 8
    Width = 217
    Height = 233
    Lines.Strings = ('Memo1')
    TabOrder = 0
  end
  object Button1: TButton
```

Green Apple Data Center

```
    Left = 256
    Top = 8
    Width = 75
    Height = 25
    Caption = '打开文件'
    TabOrder = 1
  end
  object Button2: TButton
    Left = 256
    Top = 56
    Width = 75
    Height = 25
    Caption = '存储文件'
    TabOrder = 2
  end
  object Button3: TButton
    Left = 256
    Top = 112
    Width = 75
    Height = 25
    Caption = '颜色控制'
    TabOrder = 3
  end
  object Button4: TButton
    Left = 256
    Top = 168
    Width = 75
    Height = 25
    Caption = '字体设置'
    TabOrder = 4
  end
  object Button5: TButton
    Left = 256
    Top = 216
    Width = 75
    Height = 33
    Caption = '退出'
    TabOrder = 5
  end
  object OpenDialog1: TOpenDialog
    Left = 80
    Top = 80
  end
  object SaveDialog1: TSaveDialog
    Left = 112
    Top = 104
```

Green Apple Data Center

```
    end
    object FontDialog1: TFontDialog
      Font.Charset = DEFAULT_CHARSET
      Font.Color = clWindowText
      Font.Height = -11
      Font.Name = 'MS Sans Serif'
      Font.Style = []
      MinFontSize = 0
      MaxFontSize = 0
      Left = 192
      Top = 160
    end
    object ColorDialog1: TColorDialog
      Ctl3D = True
      Left = 152
      Top = 136
    end
  end
```

2．程序的初始化

程序的初始化过程，也就是对窗体 FormCreate()**事件的初始化。**

在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()**事件的过程处理代码中，并且添加如下所示代码：**

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Memo1.Clear;
//清空文本框
form1.OpenDialog1.InitialDir:='c:\windows';
//设置缺省文件目录
form1.OpenDialog1.Filter:='All Files(*.*)|*.*|Text Files(*.txt)|*.txt';
//设置文件过滤条件
form1.Button2.Enabled:=false;
//设置按钮有效状态
form1.FontDialog1.Font:=form1.Memo1.Font;
form1.ColorDialog1.Color:=form1.Memo1.Color;
//对话框的初始化
end;
```

在程序运行的初期，程序首先清空文本框中的文本显示内容，这为用户输入文字和打开文件作好了准备工作，由于程序刚刚开始运行时，还没有打开文件，所以“ 存储文件 ”按钮应该处于无效的状态，这是通过下面的语句来实现的：

```
form1.Button2.Enabled:=false;
```

在程序的初始化过程中，还包括对 OpenDialog **控件的初始化，通过一条语句** form1.OpenDialog1.InitialDir:='c:\windows';**来设置控件所显示对话框的缺省路径为** c:\windows**，同时设置的文件显示过滤器为**'All Files(*.*)|*.*|Text Files(*.txt)|*.txt'**，这也就意味着，在程序运行的过程中，“ 打开 ”对话框显示的文件只有两种选择——要么显示所有的文件，要么只显示以***.txt **为后缀的文本文件。**

与 OpenDialog **控件初始化相同，**SaveDialog **控件的初始化过程也是由两条语句来实现：**

```
form1.SaveDialog1.InitialDir:='c:\windows';
```

Green Apple Data Center

form1.SaveDialog1.Filter:='All Files(*.*)|*.*|Text Files(*.txt)|*.txt';

它们的功能在 OpenDialog **控件的初始化过程中已经讲述过了。**

在窗体 FormCreate()**事件的最后通过两条语句实现：**

form1.FontDialog1.Font: =form1.Memo1.Font;

form1.ColorDialog1.Color:=form1.Memo1.Color;

上述两条语句完成对“字体”对话框和“颜色”对话框的初始化，把它们的初始状态设置与 Memo **控件相一致。**

如图 6-9 **所示，为初始化完毕的对话框和窗体。**



图 6-9　**初始化后的窗体及对话框**

至此，程序的初始化工作就完成了。

3．**响应“打开文件”按钮**

在程序的设计阶段，用鼠标的左键双击窗体上的“打开文件”按钮，在弹出的如图 6-10 **所示代码窗口的左侧子窗口中选择“** Tform1 **”/“** Published **”/“** Button1Click **”选项，在代码窗口的右侧子窗口中就会显示出控件“打开文件”的代码响应过程。**

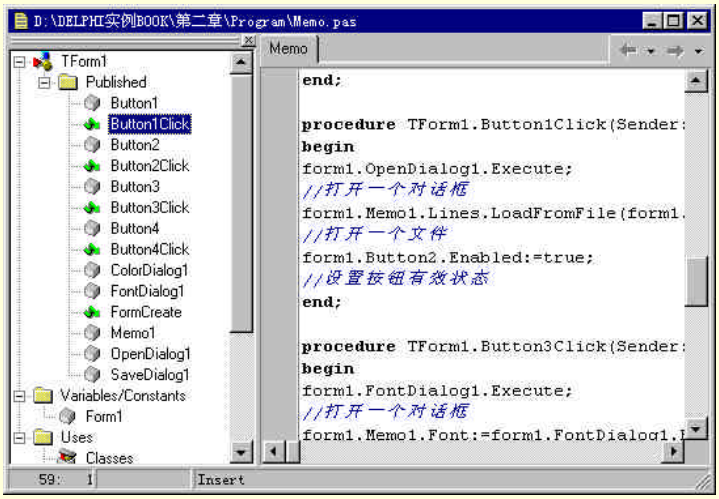


图 6-10　**弹出的代码窗口**

在其中添加如下所示的按钮响应代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
form1.OpenDialog1.Execute;
//打开一个对话框
form1.Memo1.Lines.LoadFromFile(form1.opendialog1.filename);
```

Green Apple Data Center

```
//打开一个文件
form1.Button2.Enabled:=true;
//设置按钮有效状态
end;
```

程序说明：

在所添加的代码段中，特别值得注意的是语句 form1.OpenDialog1.Execute; **和语句** form1.Memo1.Lines.LoadFromFile(form1.opendialog1.filename);**，其中前者的作用是显示一个“打开”对话框，用户可以在其中选择一个文本文件，后者的作用是打开用户在“打开”对话框中所选择的文件。**

当用户打开一个文本文件后，窗体中的“存储文件”按钮就应该变为有效的状态，这一功能是通过语句 form1.Button2.Enabled:=true;**来实现的。**

一个典型的打开文件的对话框如图 6-11 **所示。**

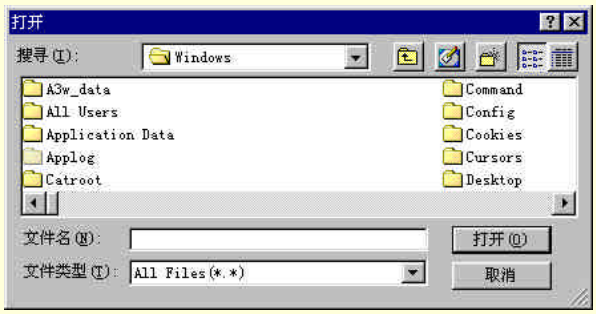


图 6-11 **“打开”对话框**

相应的，在“存储文件”按钮的事件响应中添加的代码如下所示：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
form1.SaveDialog1.Execute;
form1.Memo1.Lines.SaveToFile(form1.savedialog1.FileName);   //保存文件
end;
```

4. **响应“字体设置”和“颜色设置”按钮**

在窗体中放置有一个 FontDialog **控件和一个** ColorDialog **控件，它们的作用是分别为文本框控件设置显示字体和背景色，下面以** FontDialog **控件为例来说明在程序中如何添加响应属性设置按钮事件的代码。**

在程序的设计阶段，用鼠标的左键双击控件 FontDialog1**，在屏幕上就会弹出一个代码窗口，把光标移动到代码窗口中，并且添加如下所示的按钮事件响应代码：**

```
procedure TForm1.Button3Click(Sender: TObject);
begin
form1.FontDialog1.Execute;   //打开一个对话框
form1.Memo1.Font:=form1.FontDialog1.Font;   //设置控件中文本显示字体
end;
```

在 FontDialog **控件的响应代码中，语句** form1.FontDialog1.Execute;**的作用是显示一个对话框，用户可以在其中为** Memo **控件中的文本设置字体、大小、风格以及颜色等，而** form1.Memo1.Font:=form1.FontDialog1.Font; **语句的作用是将用户在“字体”对话框中的选择传送给** Memo **控件。**

5. **运行程序**

做完以上的工作后，选择菜单“File**”中的“**Save All**”选项，在弹出的对话框中选择合适的文件名保存文件（这里保存为项目** App2_2**），然后按键盘上的功能键** F9 **运行程序。**

首先用鼠标的左键单击按钮“打开文件”，在屏幕上就会弹出一个对话框，提示用户选择一个有效的文本文件，单击“打开”按钮，在 Memo **控件中就会显示出文本文件中的内容，如图** 6-12 **所示。**

Green Apple Data Center



图 6-12 一个打开的文本文件

在程序运行的过程中，用户可以通过单击“颜色设置”按钮和“字体设置”按钮来改变系统的设置。如图 6-13 所示为改变文本的颜色控制效果。

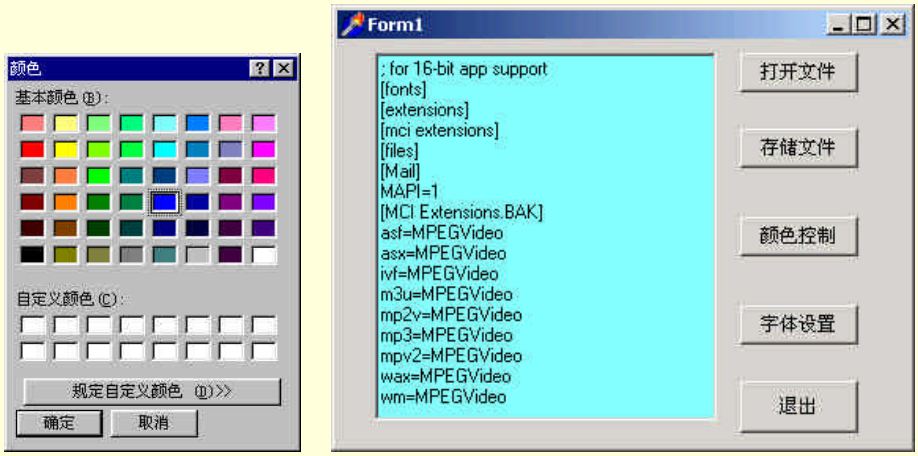


图 6-13 改变文本颜色效果

编辑文本后，单击“存储文件”按钮，在窗体上就会弹出一个如图 6-14 所示的对话框。

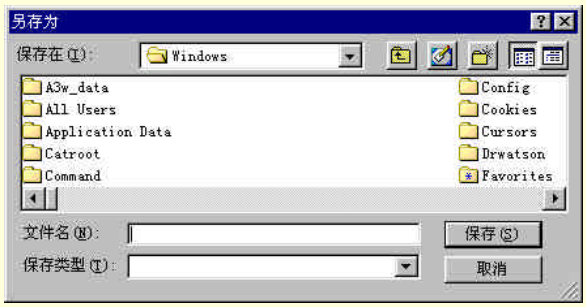


图 6-14 “另存为...”对话框

在“另存为 ...”对话框中选择一个有效的路径和文件名，单击“保存”按钮就完成了文本文件的保存工作。

6.4 Radiogroup 控件和 Checkbox 控件

在 Delphi 5 中 RadioButton 控件经常用于在一组选项中达到“多选一”的目的，而 CheckBox 控件经常用于同时用于“多选多”的控制。一个典型的利用选项控件来设置参数的窗口如图 6-15 所示。

Green Apple Data Center

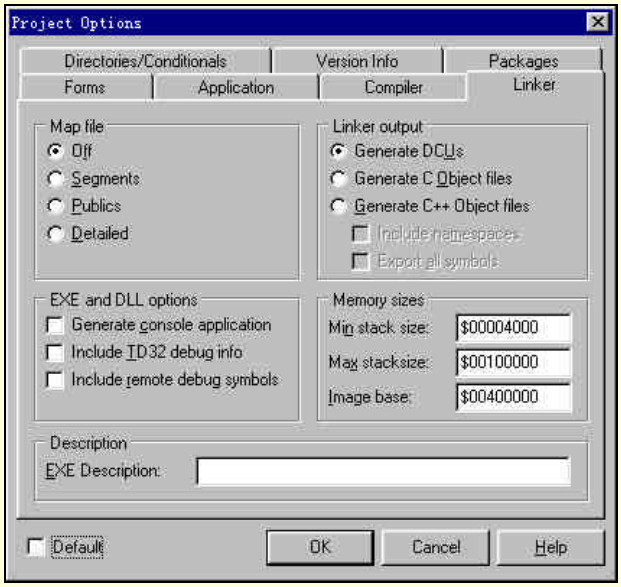


图 6-15　一个典型的选项窗口

下面我们就以一个“四则运算器”的设计实例，来说明如何在程序中利用 RadioButton 控件和 CheckBox 控件。示例程序在运行的过程中能够动态的改变中四则运算的执行方式和执行条件（如是否有余数、是否动态清空），具体的程序设计步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Standard 选项后，在 Memo 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 控件，接着向窗体上添加一个 GroupBox 控件、两个 CheckBox 控件、一个 Radiogroup 和三个 Edit 控件及对应的标签控件，它们的作用如下所示：

- Edit **控件：在程序运行的过程中接收用户的输入数值和显示的程序运行结果；**
- GroupBox **控件：在程序的设计和运行阶段将 CheckBox 控件和 RadioButton 控件分成两个组；**
- CheckBox **控件：在程序运行过程中，用户可以通过单击 CheckBox 控件来选择四则运算的执行条件；**
- Radiogroup **控件：在程序运行过程中，用户可以通过单击 Radiogroup 控件的不同选项来选择执行哪种四则运算方式。**

添加控件后的窗体如图 6-16 所示。



图 6-16　添加控件后的窗体

其中各个控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 195
  Top = 180
```

Green Apple Data Center

```
Width = 418
Height = 259
ActiveControl = Edit1
Caption = '四则运算器'
Color = clBtnFace
Font.Charset = DEFAULT_CHARSET
Font.Color = clWindowText
Font.Height = -11
Font.Name = 'MS Sans Serif'
Font.Style = []
OldCreateOrder = False
PixelsPerInch = 96
TextHeight = 13
object Label1: TLabel
  Left = 154
  Top = 28
  Width = 39
  Height = 13
  Caption = '操作数 1'
end
object Label2: TLabel
  Left = 146
  Top = 67
  Width = 39
  Height = 13
  Caption = '操作数 2'
end
object Label3: TLabel
  Left = 193
  Top = 91
  Width = 129
  Height = 13
  Caption = '----------'
end
object Label4: TLabel
  Left = 154
  Top = 111
  Width = 22
  Height = 13
  Caption = '结果'
end
object RadioGroup1: TRadioGroup
  Left = 26
  Top = 20
  Width = 103
  Height = 125
```

Green Apple Data Center

```
    Caption = '请选择运算方法'
    ItemIndex = 0
    Items.Strings = (
      '加'
      '减'
      '乘'
      '除')
    TabOrder = 0
  end
  object Edit1: TEdit
    Left = 206
    Top = 28
    Width = 85
    Height = 21
    DragCursor = crDefault
    TabOrder = 1
    OnEnter = Edit1Enter
    OnKeyPress = Edit1KeyPress
  end
  object Edit2: TEdit
    Left = 206
    Top = 67
    Width = 85
    Height = 21
    TabOrder = 2
    OnEnter = Edit2Enter
    OnKeyPress = Edit2KeyPress
  end
  object Edit3: TEdit
    Left = 206
    Top = 111
    Width = 85
    Height = 21
    TabOrder = 3
  end
  object Button1: TButton
    Left = 303
    Top = 25
    Width = 65
    Height = 26
    Caption = '运算'
    TabOrder = 4
    OnClick = Button1Click
  end
  object GroupBox1: TGroupBox
    Left = 26
```

Green Apple Data Center

```
      Top = 158
      Width = 335
      Height = 43
      Caption = '功能选择'
      TabOrder = 5
      object CheckBox1: TCheckBox
        Left = 37
        Top = 20
        Width = 111
        Height = 13
        Caption = '进入输入框时清空'
        TabOrder = 0
      end
      object CheckBox2: TCheckBox
        Left = 191
        Top = 20
        Width = 118
        Height = 13
        Caption = '除法显示余数'
        TabOrder = 1
      end
    end
    object Button2: TButton
      Left = 303
      Top = 107
      Width = 65
      Height = 33
      Caption = '退出'
      TabOrder = 6
      OnClick = Button2Click
    end
  end
```

2．程序的初始化

程序的初始化过程，也就是对窗体 FormCreate()事件的初始化。

在程序设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到 FormCreate()事件的过程处理代码中，并且添加如下所示代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
radiogroup1.itemindex:=1;
//设置 radiogroup 控件初始状态
checkbox1.Checked:=false;
checkbox2.Checked:=false;
//设置 checkbox 控件初始状态
end;
```

在程序的初始化代码中，首先通过 memo1.ImeName:='';语句设置文本框中的输入方法为英文输入方法，然后设置四个 radiobutton 控件都处于未选中的状态，同时设置了三个 checkbox 控件也处于未选中的状态。

Green Apple Data Center

3. 添加事件代码

在程序的设计阶段，用鼠标的左键双击窗体上的 Button1 控件（即“运算”按钮），在代码窗口的右侧子窗口中就会显示出控件 Button1 的代码响应过程。

```
procedure TForm1.Button1Click(Sender: TObject);
var
   x,y,z:integer;
begin
     if editempty(edit1) or editempty(edit2) then
     exit;
     x:=strtoint(edit1.text);
     y:=strtoint(edit2.text);
     z:=0;
     if (radiogroup1.itemindex=3) then
      if y=0 then
      begin
      showmessage('除数为零');
           //显示错误信息提示对话框
      exit;
      end;
     case radiogroup1.itemindex of
      0:z:=x+y;
      1:z:=x-y;
      2:z:=x*y;
      3:z:=x div y;
     end;
     //根据 radiogroup 的选项不同，响应不同的四则运算
  if checkbox2.checked and (radiogroup1.itemindex=3) then
        begin
        if(x mod y<>0) then
        edit3.text:=inttostr(z)+'余'+inttostr( x mod y);
        end
       else
        edit3.text:=inttostr(z);
     //根据 checkbox 的选项不同，响应不同的四则运算附加条件
end;
```

在程序运行的过程中，当用户单击“运算”选项按钮时，就会激活 ButtonClick()事件，然后程序就会自动的判断控件 Radiogroup 中哪个选项（加、减、乘、除）处于选中的状态，如果处于选中的状态，则执行该运算。

提示：

- 由于篇幅关系，请读者按照后面的源程序添加完其他的代码。

4. 运行程序

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件，然后按键盘上的功能键 F9 运行程序，程序运行的初始画面如图 6-17 所示。

Green Apple Data Center



图 6-17　程序运行的初始画面

在如图 6-17 所示的程序运行画面中，用鼠标的左键选中四则运算方式为“除”，保持“功能选择”中的两个条件都不被选中，然后在数值输入框中输入操作数 1 和操作数 2，假设我们这里分别输入 100 和 9，执行运算的结果如图 6-18 所示。



图 6-18　程序运行结果

现在，如果用户再重新将“功能选择”栏中的两项都选中，就会发现：一旦鼠标进入数值输入框中的数据就会清空，作四则运算的除法运算时，输出结果还会显示余数。某一次运算的结果如图 6-19 所示，这里除数为 11，被除数 1000。



图 6-19　程序运行结果

附程序的完整源代码如下所示：

Green Apple Data Center

程序清单

```
unit sizeyunsuan;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    RadioGroup1: TRadioGroup;
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Edit2: TEdit;
    Label3: TLabel;
    Label4: TLabel;
    Edit3: TEdit;
    Button1: TButton;
    GroupBox1: TGroupBox;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Edit1Enter(Sender: TObject);
    procedure Edit2Enter(Sender: TObject);
    procedure Edit1KeyPress(Sender: TObject; var Key: Char);
    procedure Edit2KeyPress(Sender: TObject; var Key: Char);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

//以下是一个自定义函数，用于文本框的清空
```

Green Apple Data Center

```
function editempty(editbox:Tedit):boolean;
begin
if editbox.text='' then
begin
editbox.color:=clred;
editbox.text:='请输入';
result:=true;
end
else
begin
editbox.Color:=clwindow;
result:=false;
end
end;

procedure TForm1.Button1Click(Sender: TObject);
var
   x,y,z:integer;
begin
     if editempty(edit1) or editempty(edit2) then
     exit;
     x:=strtoint(edit1.text);
     y:=strtoint(edit2.text);
     z:=0;
     if (radiogroup1.itemindex=3) then
       if y=0 then
       begin
       showmessage('除数为零');
       exit;
       end;
     case radiogroup1.itemindex of
       0:z:=x+y;
       1:z:=x-y;
       2:z:=x*y;
       3:z:=x div y;
     end;
       if checkbox2.checked and (radiogroup1.itemindex=3) then
         begin
         if(x mod y<>0) then
         edit3.text:=inttostr(z)+'余'+inttostr( x mod y);
         end
       else
         edit3.text:=inttostr(z);
end;

procedure TForm1.Button2Click(Sender: TObject);
```

Green Apple Data Center

```
begin
close();
end;

procedure TForm1.Edit1Enter(Sender: TObject);
begin
if checkbox1.checked then
begin
edit3.text:='';
edit1.text:='';
end;
end;

procedure TForm1.Edit2Enter(Sender: TObject);
begin
if checkbox1.checked then
begin
edit3.text:='';
edit2.text:='';
end;
end;

procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
if not(key in['0'..'9']) then
key:=chr(0);
end;

procedure TForm1.Edit2KeyPress(Sender: TObject; var Key: Char);
begin
if not(key in['0'..'9']) then
key:=chr(0);
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
radiogroup1.Itemindex:=1;
radiogroup1.itemindex:=1;
//设置 radiogroup 控件初始状态
checkbox1.Checked:=false;
checkbox2.Checked:=false;
//设置 checkbox 控件初始状态
end;
end.
```

Green Apple Data Center

6.5 DirectoryOutline 控件和 Animate 控件

在 Windows 98 或者 Windows 2000 操作系统中，开始菜单中都有一个查找文件的程序，利用这个应用程序可以在本地机上查询以匹配符为条件的文件。在 Delphi 中，利用 VCL 控件也可以制作类似的文件查询程序。

下面就以 DirectoryOutline 控件和动态 Animate 控件为例来说明在 Delphi 中如何制作一个文件查询程序，具体的程序制作步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Standard 选项后，在 Button 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件。

添加按钮控件后，把鼠标移动到控件工具栏上的其它选项上，向窗体上添加一个 DirectoryOutline 控件、一个 Animate 控件、一个 DriveComboBox 控件和一个 Edit 控件，各个控件的功能在后面将予以详细的介绍。

添加控件后的窗体如图 6-20 所示。



图 6-20 添加控件后的窗体

其中窗体及控件的属性设置如下所示：

```
object Form1: TForm1
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  OldCreateOrder = False
  PixelsPerInch = 96
  object Label1: TLabel
    Left = 248
    Top = 32
    Width = 192
    Height = 29
    Caption = '请输入查找文件名'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -24
    Font.Name = 'MS Sans Serif'
    Font.Style = []
```

Green Apple Data Center

```
    ParentFont = False
  end
  object DriveComboBox1: TDriveComboBox
    Left = 24
    Top = 8
    Width = 209
    Height = 19
    TabOrder = 0
  end
  object Edit1: TEdit
    Left = 248
    Top = 80
    Width = 193
    Height = 21
    TabOrder = 1
    Text = 'Edit1'
  end
  object DirectoryOutline1: TDirectoryOutline
    Left = 24
    Top = 40
    Width = 209
    Height = 289
    ItemHeight = 13
    Options = [ooDrawFocusRect]
    PictureLeaf.Data = {·····}
    TabOrder = 2
    Data = {10}
  end
  object Animate1: TAnimate
    Left = 272
    Top = 200
    Width = 48
    Height = 50
    Active = False
    CommonAVI = aviFindFile
    ParentColor = False
    ParentShowHint = False
    ShowHint = False
    StopFrame = 23
    Timers = True
  end
  object Button1: TButton
    Left = 264
    Top = 136
    Width = 75
    Height = 25
```

Green Apple Data Center

```
    Caption = '开始查找'
    TabOrder = 4
  end
end
```

通过以上的属性设置，控件具有如下所示的特性：

- 文本控件中显示为“ *.* ”；
- 按钮控件上显示为“ 开始查找 ”；
- 在程序的设计阶段就可以看见 Animate 控件的动画。

2．程序的初始化

在程序的设计过程中，用鼠标的左键双击窗体中的空白处，在屏幕上就会弹出一个代码窗口，光标的缺省位置是在窗体 FormCreate()事件的处理过程中，在光标的当前位置添加如下代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.DriveComboBox1.Drive:='c';
//初始化驱动器
form1.DirectoryOutline1.Directory:='c:\windows';
//初始化工作目录
form1.Animate1.CommonAVI:=avifindfile;
//初始化播放动画；
form1.Animate1.Active:=false;
//停止动画
form1.Edit1.Clear;
//清空文本框
form1.Button1.Enabled:=false;
//设置按钮的有效状态
end;
```

其中各个控件的功能如下所示：

- 文本框控件：用户输入文件查询条件的容器；
- 按钮控件：产生查询动作；
- Animate 控件：放映查找文件动画；
- DriveComboBox 控件：选择查找文件的驱动器；
- DirectoryOutline 控件：选择查找文件的路径。

3．响应文本框事件

在程序的初始化代码中，通过语句 form1.Edit1.Clear;清空了文本框中的内容，同时通过 form1.Button1.Enabled:=false;设置按钮的有效状态为无效状态，如果在程序的运行过程中用户改变了文本框中的内容，就应该使得按钮控件处于有效的状态。

为此，在程序的设计阶段用鼠标的左键双击文本框控件，在屏幕上就会弹出一个代码窗口，在其中可以添加如下所示的响应文本框事件的代码：

```
procedure TForm1.Edit1Change(Sender: TObject);
begin
form1.Button1.Enabled:=true;
//设置按钮的有效状态
end;
```

4．设置文件查找路径

查找文件之前要为程序提供一个查找的路径，即文件所在的驱动器和文件目录，在本程序中文件的查找路径是通过 DirectoryOutline 控件所提供的，但是仅仅一个 DirectoryOutline 控件是不够的，在程序运行的过程中，

Green Apple Data Center

DirectoryOutline 控件只能够提供当前工作驱动器下的文件目录，用户只有自己在程序中手动的调节。

单个 DirectoryOutline 控件的工作情形如图 6-21 所示。

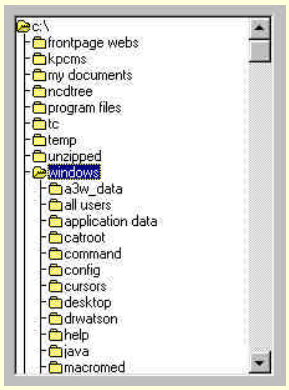


图 6-21　DirectoryOutline 控件的工作情形

为了能够在程序的运行的过程中选择更多的驱动器，在窗体上特地放置了一个 DriveComboBox 控件，一个 DriveComboBox 控件在程序运行的过程中可以为用户提供用户计算机上的所有有效驱动器，如图 6-22 所示。

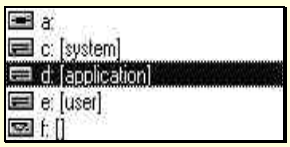


图 6-22　程序运行中的 DriveComboBox 控件

为了使 DriveComboBox 控件与 DirectoryOutline 控件能够在程序运行的过程中相互协调的工作，在程序的设计阶段双击窗体上的 DriveComboBox 控件，在屏幕上就会弹出一个代码窗口，把光标移动到 DriveComboBox 控件的 DriveComboBox1Change()事件处理过程中，并且添加如下所示的程序代码：

```
procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
form1.DirectoryOutline1.Drive:=form1.DriveComboBox1.Drive;
//显示指定驱动器下的目录
end;
```

执行以上的代码段，在程序的运行过程中，单击 DriveComboBox 控件，可以获得当前计算机上的所有有效驱动器，同时在 DirectoryOutline 控件中会显示出当前驱动器下的所有文件目录。

5．Animate 控件

窗体上的 Animate 控件用于在文件查找的过程中显示一个动画，这个动画就是大家非常熟悉的“查找”动画。在程序设计的过程中，我们可以通过设置 Animate 控件的 CommonAVI 属性来获得诸如“查找”等八个系统动画，如图 6-23 所示为系统 Findfile 动画。



图 6-23 “查找文件”动画

当然也可以在程序中通过代码指定文件的 Filename 属性来达到同样的效果，如下所示：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Animate1.FileName:='d:\delphi\demos\cool.avi';
//设置动画文件
end;
```

Green Apple Data Center

6. 响应“查询”动作

程序中文件的“查询”动作是通过按钮控件来实现的，但是实际上，按钮控件只是提供了一个动作的框架，实际的工作是通过两个函数 FileSearch()和 MessageDlg()函数来完成的，它们的语法结构如下所示：

```
function FileSearch(const Name, DirList: string): string;
function MessageDlg(const Msg: string; AType: TMsgDlgType; AButtons:
                    TMsgDlgButtons; HelpCtx: Longint): Word;
```

函数说明：

其中函数 FileSearch()的作用是在指定的路径下查找一个符合给定条件的文件，而 MessageDlg()函数的功能是显示一个查找结果的对话框。

为了在程序中实现“查询”的功能，在程序的设计阶段，用鼠标的左键双击“开始查询”按钮，在弹出的代码窗口中添加如下所示的程序响应代码：

```
procedure TForm1.Button1Click(Sender: TObject);
var find:string;
begin
form1.Animate1.Active:=true;
find:=filesearch(form1.edit1.text,form1.DirectoryOutline1.Directory);
if find='' then
    messagedlg('未找到文件'+form1.edit1.text,mtinformation,mbyesnocancel,0)
    else
    messagedlg('找到文件'+form1.edit1.text,mtinformation,mbyesnocancel,0);
form1.Animate1.Active:=false;
end;
```

在程序的运行过程中，当用户在文本输入框中输入一个有效的文件名后，“开始查询”按钮就会变为有效的状态，单击按钮控件，就会自动的激活 Button1Click()事件，程序首先定义了一个字符串变量 filnd，在其中存储文件查找结果。

然后通过语句 find:=filesearch(form1.edit1.text,form1.DirectoryOutline1.Directory);来查找指定路径下面的文件，如果存在符合条件的文件，就会弹出一个对话框信息，告诉用户查找的结果。

7. 运行程序

做完以上各步的工作后，选择菜单 File 中的 Save All 选项保存文件，然后按键盘上的功能键 F9 运行程序，程序运行的初始画面如图 6-24 所示。



图 6-24 程序运行的初始画面

在程序运行的画面中，选择驱动器为 C，选择工作目录为 C:\windows，在文本框中输入查找的文件为 win.ini，单击“开始查询”按钮，结果如图 6-25 所示。

Green Apple Data Center

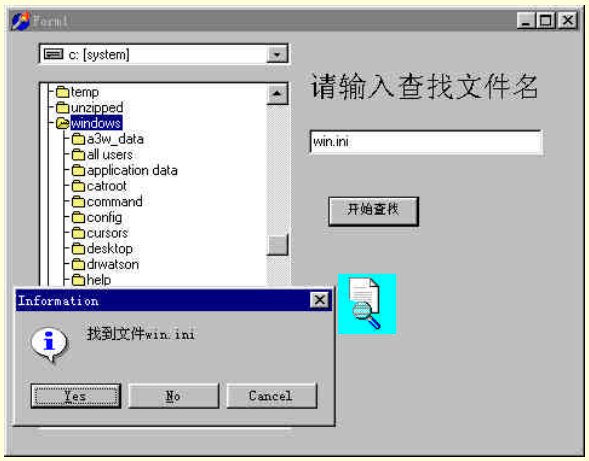


图 6-25　查询文件已找到

而如果输入一个本来不存在的文件，比如我们随便输入的 temp，就会看到如图 6-26 所示的情形。



图 6-26　查询文件未果

程序的完整源代码如下所示：

程序清单

```
unit find;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ComCtrls, Grids, Outline, DirOutln, FileCtrl;

type
  TForm1 = class(TForm)
    DriveComboBox1: TDriveComboBox;
    Label1: TLabel;
    Edit1: TEdit;
    DirectoryOutline1: TDirectoryOutline;
    Animate1: TAnimate;
```

Green Apple Data Center

```
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure DriveComboBox1Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.DriveComboBox1.Drive:='c';
//初始化驱动器
form1.DirectoryOutline1.Directory:='c:\windows';
//初始化工作目录
form1.Animate1.CommonAVI:=avifindfile;
//初始化播放动画；
form1.Animate1.Active:=false;
//停止动画
form1.Edit1.Clear;
//清空文本框
form1.Button1.Enabled:=false;
//设置按钮的有效状态
end;

procedure TForm1.Edit1Change(Sender: TObject);
begin
form1.Button1.Enabled:=true;
//设置按钮的有效状态
end;

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
form1.DirectoryOutline1.Drive:=form1.DriveComboBox1.Drive;
//显示指定驱动器下的目录
end;

procedure TForm1.Button1Click(Sender: TObject);
```

Green Apple Data Center

```
var find:string;
begin
form1.Animate1.Active:=true;
find:=filesearch(form1.edit1.text,form1.DirectoryOutline1.Directory);
if find='' then
    messagedlg('未找到文件'+form1.edit1.text,mtinformation,mbyesnocancel,0)
    else
    messagedlg('找到文件'+form1.edit1.text,mtinformation,mbyesnocancel,0);
form1.Animate1.Active:=false;
end;
end.
```

6.6 小 结

在本章中，通过几个示例程序的具体制作，向用户说明了在 Delphi 中几个基本控件——Button 控件、Edit 控件、Label 控件、Memo 控件、CheckBox 控件、RadioButton 控件和 Animate 控件的使用方法，熟练的运用这些控件，是设计一个好的应用程序的基础，希望读者能够仔细的体会本章示例程序中对基本控件的处理过程，并且为以后的学习打下良好的基础。

Green Apple Data Center

第七章　管理文件、目录和驱动器

所谓的格式文件是指存放在外部存储介质上的数据和程序等，在 Windows 的操作环境中，文件是一个非常重要的概念。

为了方便用户存取和操作的方便，在 Delphi 中带有强大的文件管理功能，在其中用户可以通过调用文件处理函数来对文件和目录进行各种各样的处理和操作，如创建目录，打开和删除文件等操作。

7.1　各种控件简介

在本章中，将讨论以下几个问题：

- 如何使用常用的文件类 ActiveX 控件
- 如何利用文件类控件来设计文件系统的界面
- 利用文件类的函数来进行文件内部的操作

下面首先来介绍一下文件类 ActiveX 控件的使用方法。

Delphi 中的文件类控件包括 DirectoryListBox 控件（目录列表框控件）、FileListBox 控件（文件列表框控件）、DriveComboBox 控件（驱动器组合列表框控件）和 FilterComboBox 控件（过滤组合列表框控件）四种。

如图 7-1 所示即为一个装有以上四种文件类控件的窗体。

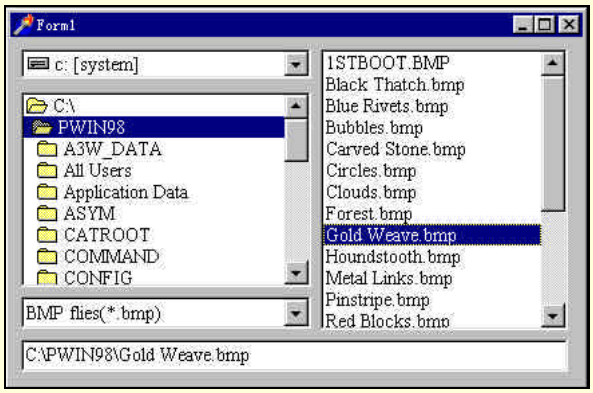


图 7-1　放置文件类控件后的窗体

在图 7-1 中，这四个文件类控件之间的关系必须是协调的，如 DriveComboBox 控件中显示的是 C 驱动器，则 DirectoryListBox 控件也必须显示的是“ C:\Pwin98 ”， FileListBox 控件中显示的是“ C：\Pwin98 ”中的 BMP 文件内容。实际上，实现这些协调，需要用户编制程序才能实现，下面我们就来一起学习。

7.1.1　驱动器控件

DriveComboBox 控件（驱动器组合列表框控件）的功能是显示当前计算机中全部有效的驱动器，以供用户选择和操作。

DriveComboBox 控件的属性在这里就不一一的加以介绍了，下面以一个示例来说明 DriveComboBox 控件和它的属性的用法，具体步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Win32 ”选项后，在“ DriveComboBox 控件 ”的图标上双击鼠标的左

Green Apple Data Center

键，这时空白的窗体上就会出现一个 DriveComboBox 控件，如图 7-2 所示。

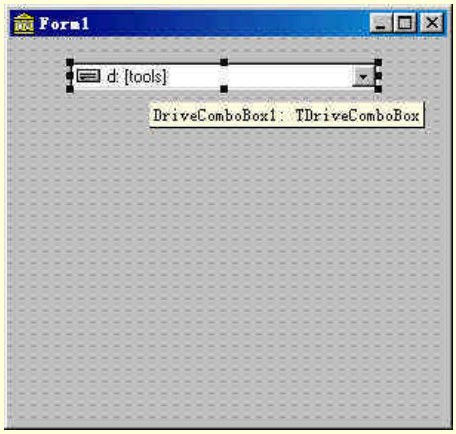


图 7-2　添加控件后的窗体

2. 添加初始化代码

添加控件后，用鼠标的左键在窗体上的空白处双击，把光标移动到代码窗口中的适当位置，找到需要添加代码的事件——窗体的 FormCreate()事件，在其中添加如下代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
drivecombobox1.Drive:='d';
//设置驱动器为 D 盘
drivecombobox1.TextCase:=tcUpperCase;
//设置显示的字母为大写字母
end;
```

程序说明：

DriveComboBox 控件的 Drive 用于设置控件显示的驱动器的盘符，所以语句 drivecombobox1.Drive:='d';的作用是设置控件显示的驱动器为 D 盘；

而 TextCase 属性用于设置 DriveComboBox 控件中的字符显示形式，所以语句 drivecombobox1.TextCase:=tcUpperCase;的作用是设置控件显示的字符为大写字符。

在以上的程序段中，我们通过语句 drivecombobox1.TextCase:=tcUpperCase;来设置 DriveComboBox 控件中显示的字符为大写字符，那么可不可以通过程序来设置 DriveComboBox 控件中显示为小写字符呢？

提示：

> 与设置显示字符为大写字符一样，设置 DriveComboBox 控件中显示为小写字符的语句为 drivecombobox1.TextCase:=tcLowerCase;。

3. 运行程序

做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序，程序的运行结果如图 7-3 所示。

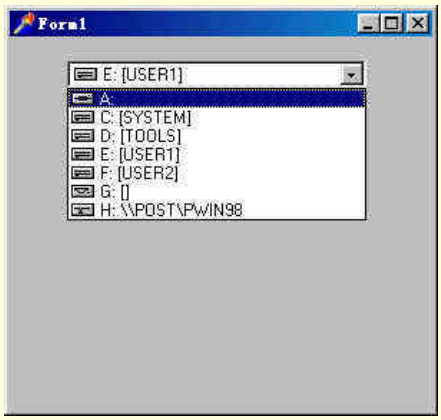


图 7-3　程序运行结果

Green Apple Data Center

7.1.2 目录控件

DirectoryListBox 控件（目录列表框控件）的功能是显示当前计算机有效驱动器中当前驱动器的树形目录结构，以供用户选择和操作，用户在选择的过程中可以在同一个驱动器的不同目录之间进行切换等操作。

DirectoryListBox 控件的属性在这里就不一一的加以介绍了，如果读者有什么不清楚的地方，请参考 DriveComboBox 控件即可。

下面以一个示例来说明 DirectoryListBox 控件控件和它的属性的用法，具体步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Win3.1 ”选项后，在“ DirectoryListBox 控件 ”的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 DiretoryListBox 控件，然后在控件工具栏上选择‘ Standard ’选项，在‘ Label 控件 ”的图标上双击鼠标的左键，在窗体上就会添加一个 Label 控件。

添加控件后的窗体如图 7-4 所示。

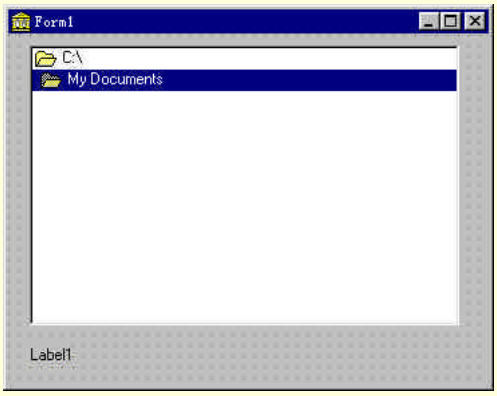


图 7-4 添加控件后的窗体

2. 添加代码

当然，控件和窗体的属性设置不但可以在程序的设计过程中由用户手动设置，还可以在程序的运行过程中有程序控制，如以上的两个控件的属性设置可以通过以下的程序段来实现。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
chdir('c:\mydocument');
//设置系统的缺省路径为 c:\mydocument
directorylistbox1.Align:=alTop;
//DirectoryListBox 控件始终位于窗体的顶部
directorylistbox1.DirLabel:=label1;
//与 DirectoryListBox1 相关联的控件为 label1
end;
end.
```

3. 运行程序

做完以上的工作后，按键盘上的功能键 F9 运行程序，程序的运行结果如图 7-5 所示。

Green Apple Data Center

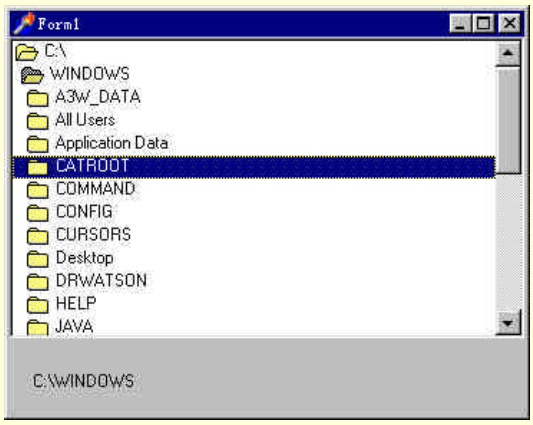


图 7-5　程序运行结果

在程序的运行过程中，可以选择不同的目录，同时在 Label 控件中会显示出用户在程序运行过程中所选择的路径。

7.1.3　文件列表控件

FileListBox 控件（文件列表框控件）的功能是显示当前路径中符合条件的所有文件列表，以供用户选择和操作，用户在选择的过程中可以在同一个路径中的不同文件之间进行切换等操作。

FileListBox 控件的属性在这里就不一一的加以介绍了，如果读者有什么不清楚的地方，请参考有关的参考手册，下面以一个示例来说明 FileListBox 控件和它的属性的用法，另外还介绍了与 FileListBox 控件相关联的 Edit 控件，其中 Eidt 控件用于显示 FileListBox 控件中的文件名。

制作示例程序的具体步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单“File”中的“New Application”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 FileListBox 控件的图标、Edit 控件的图标和 Button 控件的图标，并且把它们拖放到空白的窗体上，添加控件后的窗体如图 7-6 所示。



图 7-6　添加控件后的窗体

2．添加程序代码

在本程序中需要添加代码的事件有窗体的 FormCreate()事件，按钮 Button1MouseDown()事件、Button1MouseUp()事件和 FileListBox 控件的 FileListBox1Change()事件，在这里只以窗体的 FormCreate()事件为例来说明添加代码的过程，其它事件的响应代码请读者参见附后的源程序。

在窗体的事件列表框中选择 FormCreate()事件，这时在代码窗口中就会显示出 FormCreate()事件的框架，将光标定位到代码窗口中，并且移动到需要添加代码的位置，然后添加下列代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
filelistbox1.Drive:='d';
//设置驱动器
```

Green Apple Data Center

```
filelistbox1.Directory:='d:\delphi\bin';
//设置文件显示路径
filelistbox1.FileEdit:=edit1;
//设置关联控件
end;
```

窗体的 FormCreate()事件在窗体的装入的同时就会被激活，进入到窗体 FormCreate()事件的过程中，首先通过语句 filelistbox1.Drive:='d';设置系统默认的驱动器为 D 盘，然后设置文件的显示路径为 d:\delphi\bin，最后通过语句 filelistbox1.FileEdit:=edit1;来设置 Edit1 控件为 FileListBox1 控件的关联控件。

3．运行程序

做完以上的工作后，按键盘上的功能键 F9 运行程序，程序的运行结果如图 7-7 所示。

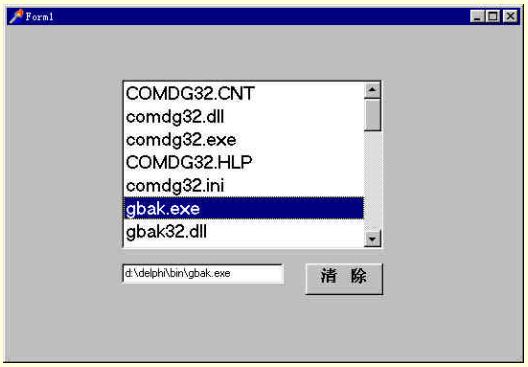


图 7-7　程序运行结果

在程序的运行过程中，当用户在 FileListBox1 控件中选择一个有效的文件时，在 Edit1 控件中就会显示出选中文件的路径和文件名，如果用户在“ 清除 ”按钮上按下鼠标的左键时，在 FileListBox1 控件中就会清除文件列表，但是只要抬起鼠标，在 FileListBox1 控件中又会重新显示指定路径下的文件列表，如图 7-8 所示。

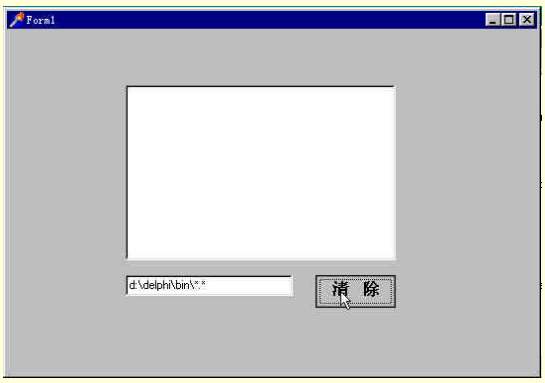


图 7-8　清除文件列表框中的内容

7.1.4　文件过滤器控件

Delphi 中常用的文件类控件中最后一种控件为 FilterComboBox 控件，作用是为用户提供从文件列表框中筛选文件的条件，它的作用决定了 FilterComboBox 控件要同其它几种文件类控件相互协调才能够完成文件的筛选功能。

FilterComboBox 控件的属性在这里就不一一的加以介绍了，如果读者有什么不清楚的地方，请参考有关的参考手册，下面以一个示例来说明 FilterComboBox 控件和它的属性的用法，另外还介绍了如何使 FilterComboBox 控件与 FileListBox 控件相互协调工作。制作示例程序的具体步骤如下所示：

1．开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项；

然后，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，选择 FileListBox 控件和 FilterComboBox 控件，

Green Apple Data Center

并且把它们拖放到空白的窗体上，添加控件后的窗体如图 7-9 所示。

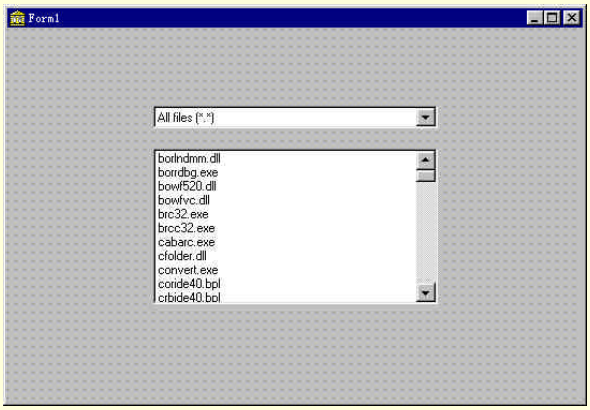


图 7-9　添加控件后的窗体

2．过滤器设置

在程序的设计阶段，用鼠标的左键单击 FilterComboBox 控件属性列表中的 Filter 属性设置框中的按钮，就会弹出一个如图 7-10 所示的过滤器设置对话框，用户可以在其中设置文件显示的过滤器。

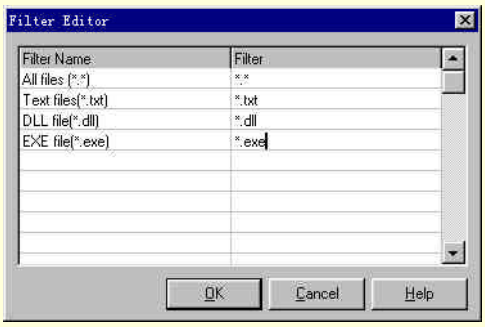


图 7-10　过滤器设置

在本程序中，过滤器的属性设置为 All files (*.*)|*.*|Text files(*.txt)|*.txt|DLL file(*.dll)|*.dll|EXE file(*.exe)|*.exe，也就是说，在程序的运行过程中，用户在显示文件类型上有四种选择：

- 显示全部文件；
- 显示以*.txt 结尾的文本文件；
- 显示以*.dll 结尾的动态链接库文件；
- 显示以*.exe 结尾的可执行文件。

3．响应窗体事件

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，在缺省的情况下，代码窗口中显示的是窗体的 FormCreate()事件，将光标移动代码窗口中，并且添加下列代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
filtercombobox1.FileList:=filelistbox1;
//设置关联控件
filelistbox1.Drive:='d';
//设置控件的驱动器
filelistbox1.Directory:='d:\delphi\bin'
//设置控件的文件路径
end;
end.
```

窗体的 FormCreate()事件在窗体的装入的同时就会被激活，进入到窗体的 FormCreate()事件的过程中，首先

Green Apple Data Center

通过语句 filtercombobox1.FileList:=filelistbox1;FileListBox1 控件为 FilterComboBox 控件的关联控件，然后设置文件列表框控件的文件路径为 d:\delphi\bin。

4. 运行程序

做完以上的工作后，按键盘上的功能键 F9 运行程序，程序的运行结果如图 7-11 所示。

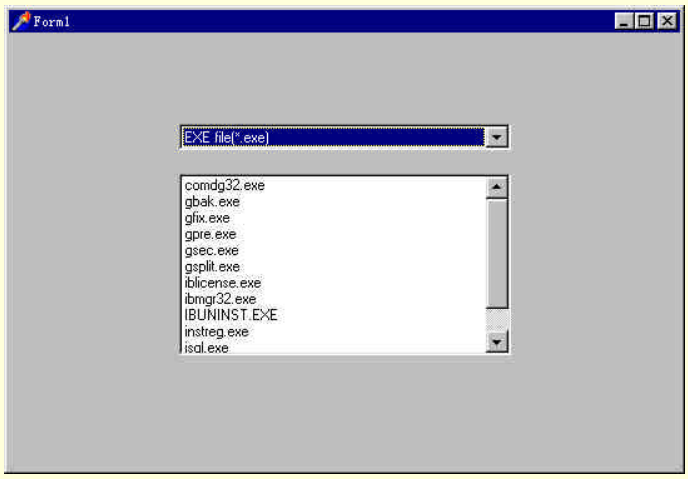


图 7-11 程序运行结果

在程序的运行过程中，用户可以选择显示文件的类型，如图 7-11 所示即为显示为以*.exe 结尾的可执行文件，当然用户也可以选择其它的文件筛选方式，如图 7-12 所示即为显示所有的以*.dll 结尾的动态链接库文件的程序运行结果。

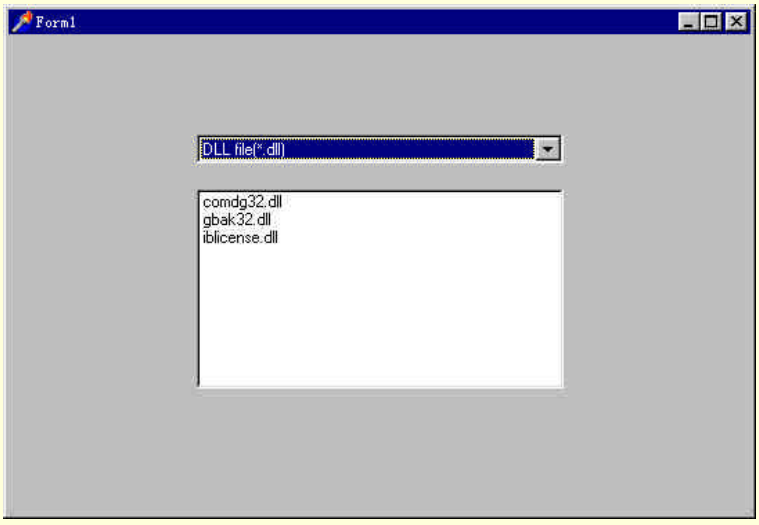


图 7-12 显示 DLL 文件

7.2 利用控件管理文件

前面分别的介绍了文件类控件中常用的四种 ActiveX 控件，但是到现在为止，各个控件之间还是独立运行，还没有使它们协调的工作，下面将通过一个综合应用的示例程序来说明如何使得四个文件类控件协调工作。

1. 开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 FileListBox 控件、Edit 控件、DriveComboBox 控件、DirectoryListBox 控件和 FilterComboBox 控件，并且把它们拖放到空白的窗体上，添加控件后的窗体如图 7-13 所示。

Green Apple Data Center

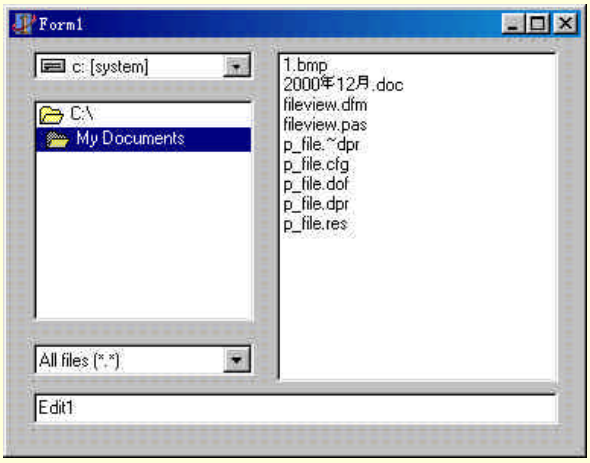


图 7-13　添加控件后的窗体

其中各个控件的作用如下所示：

- DriveComboBox 控件的作用是显示当前计算机中全部有效的驱动器，以供用户选择和操作；
- DirectoryListBox 控件的作用是显示当前计算机有效驱动器中当前驱动器的树形目录结构，以供用户选择和操作；
- FileListBox 控件的作用是显示当前路径中符合条件的所有文件列表，以供用户选择和操作；
- FilterComboBox 控件的作用是为用户提供从文件列表框中筛选文件的条件；
- Edi 控件的作用是显示选中文件的路径和文件名。

2．控件属性设置

添加控件后的工作就是设置控件的属性，在设置属性的过程中，我们要力争发挥各个控件的作用，如果能够在控件的属性设置过程中充分的发挥控件的独特作用，那么就可以减少很多编程的工作量。

在本示例程序中各个控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 384
  Height = 298
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object DriveComboBox1: TDriveComboBox
    Left = 16
    Top = 8
    Width = 145
    Height = 19
    TabOrder = 0
  end
  object DirectoryListBox1: TDirectoryListBox
```

Green Apple Data Center

```
    Left = 16
    Top = 48
    Width = 145
    Height = 145
    ItemHeight = 16
    TabOrder = 1
  end
  object FileListBox1: TFileListBox
    Left = 176
    Top = 8
    Width = 185
    Height = 217
    ItemHeight = 13
    TabOrder = 2
  end
  object FilterComboBox1: TFilterComboBox
    Left = 16
    Top = 200
    Width = 145
    Height = 21
    TabOrder = 3
  end
  object Edit1: TEdit
    Left = 16
    Top = 232
    Width = 345
    Height = 21
    TabOrder = 4
    Text = 'Edit1'
  end
end
```

3. 添加响应驱动器改变代码

如果想要在程序的运行过程中，动态的跟踪驱动器的改变，就要在 DriveComboBox **控件的** DriveComboBox1Change()**事件中添加响应驱动器改变的代码。**

在程序的设计过程中，用鼠标的左键双击控件 DriveComboBox1**，在缺省的情况下，代码窗口中显示的是** DriveComboBox **控件的** DriveComboBox1Change()**事件代码段，把光标移动到** DriveComboBox1Change()**事件的代码段中，添加如下代码：**

```
procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
form1.DirectoryListBox1.Drive:=form1.DriveComboBox1.Drive;
//设置控件 DirectoryListBox1 的驱动器
end;
```

程序说明：

当程序运行过程中，当用户改变控件 DriveComboBox1 **中的驱动器时，就会激活** DriveComboBox1 **控件的** DriveComboBox1Change()**事件，然后通过语句** form1.DirectoryList Box1.Drive:=form1.DriveComboBox1.Drive;**来动态的跟踪系统驱动器的变化。**

Green Apple Data Center

4. 添加响应目录改变代码

在程序的设计过程中，用鼠标的左键双击控件 DirectoryListBox1，在屏幕上就会弹出一个代码窗口，代码窗口中显示的是 DirectoryListBox 控件的 DirectoryListBox1Change() 事件代码段，把光标移动到 DirectoryListBox1Change()事件的代码段中，添加如下代码：

```
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
form1.FileListBox1.Directory:=form1.DirectoryListBox1.Directory;
//设置 FileListBox1 控件的目录
end;
```

5. 添加其余代码

当然，仅仅依靠前面所添加的代码是不够的，还需要添加响应建立窗体等事件的代码，如下所示：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.DriveComboBox1.Drive:='c';
//设置控件 DriveComboBox1 的初始驱动器
form1.DriveComboBox1.DirList:=form1.DirectoryListBox1;
//设置 DriveComboBox1 控件的关联控件为 DirectoryListBox1
form1.DirectoryListBox1.Directory:='c:\windows';
//设置控件 DirectoryListBox1 的初始工作目录为 c:\windows
form1.DirectoryListBox1.FileList:=form1.FileListBox1;
//设置 DirectoryListBox1 控件的关联控件为 FileListBox1
{初始化 FileListBox1 控件}
form1.FileListBox1.FileEdit:=form1.Edit1;
//设置 FileListBox1 控件的关联控件为 Edit1
form1.FilterComboBox1.FileList:=form1.FileListBox1;
//设置 FileListBox1 控件为 FilterComboBox1 控件的关联控件
end;
```

由于篇幅的限制，在这里就不对代码逐一的加以说明，详细的代码可以参看附后的源程序代码。

6. 运行程序

做完以上的工作后，存储文件（这里保存项目文件为 P_file，单元文件为 fileview）。按键盘上的功能键 F9 运行程序，程序的运行结果如图 7-14 所示，磁盘、目录和文件都能很好的配合管理。

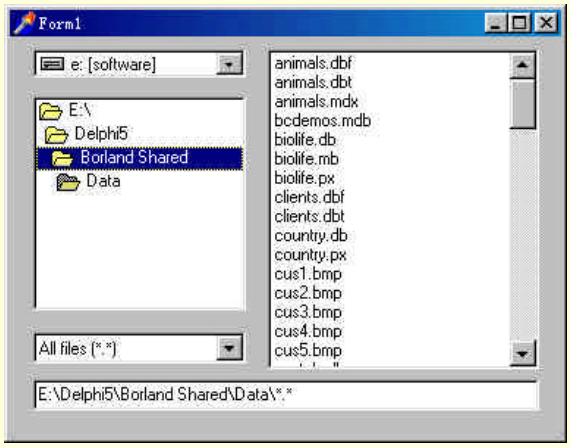


图 7-14 程序运行结果

以上程序的源程序代码如下所示：

程序清单

Green Apple Data Center

```
unit fileview;
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, FileCtrl;

type
  TForm1 = class(TForm)
    DriveComboBox1: TDriveComboBox;
    DirectoryListBox1: TDirectoryListBox;
    FileListBox1: TFileListBox;
    FilterComboBox1: TFilterComboBox;
    Edit1: TEdit;
    procedure DriveComboBox1Change(Sender: TObject);
    procedure DirectoryListBox1Change(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FileListBox1Change(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.DriveComboBox1Change(Sender: TObject);
begin
form1.DirectoryListBox1.Drive:=form1.DriveComboBox1.Drive;
//设置控件 DirectoryListBox1 的驱动器
end;

procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
form1.FileListBox1.Directory:=form1.DirectoryListBox1.Directory;
//设置 FileListBox1 控件的目录
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.DriveComboBox1.Drive:='c';
```

Green Apple Data Center

```
//设置控件 DriveComboBox1 的初始驱动器
form1.DriveComboBox1.DirList:=form1.DirectoryListBox1;
//设置 DriveComboBox1 控件的关联控件为 DirectoryListBox1
form1.DirectoryListBox1.Directory:='c:\windows';
//设置控件 DirectoryListBox1 的初始工作目录为 c:\windows
form1.DirectoryListBox1.FileList:=form1.FileListBox1;
//设置 DirectoryListBox1 控件的关联控件为 FileListBox1

{初始化 FileListBox1 控件}
form1.FileListBox1.FileEdit:=form1.Edit1;
//设置 FileListBox1 控件的关联控件为 Edit1
form1.FilterComboBox1.FileList:=form1.FileListBox1;
//设置 FileListBox1 控件为 FilterComboBox1 控件的关联控件
end;

procedure TForm1.FileListBox1Change(Sender: TObject);
begin
edit1.Text:=filelistbox1.Directory+'\'+edit1.Text;
//显示文件路径和文件名
end;
end.
```

7.3 管理目录

在 Delphi 的文件程序设计中，由于要经常的访问和处理存储在文件中的数据，所以文件类函数在程序设计中就显得十分的重要，在 Delphi 中，系统为用户提供了很多文件类的函数，其中有对文件的操作函数、对目录进行操作的函数，还有访问磁盘的函数等等，下面就对这几种函数分别的加以介绍和说明。

在 Delphi 的目录类函数中，常用的函数有 ChDir()、MkDir()和 RmDir()，它们的功能和使用方法如下。

7.3.1 ChDir 函数

ChDir()函数的功能是更改 DOS 逻辑驱动器和缺省工作目录，它的语法如下：

```
ChDir(Var S:String)
```

如果执行下面的程序段：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
ChDir('e:\user\');
//更改工作目录
end;
```

程序运行的结果是系统的工作目录更改为 e:\user。

7.3.2 MkDir 函数

MkDir()函数的功能是创建一个新的子目录，它的语法如下：

```
MkDir(Var S:String)
```

如果执行以下的程序段：

```
procedure TForm1.FormCreate(Sender: TObject);
```

Green Apple Data Center

```
begin
ChDir('c:\windows');
//更改工作目录
MkDir('test');
//在当前的工作目录下创建一个子目录
end;
end.
```

程序运行的结果是系统在 c:\windows 目录中创建一个工作目录 test。

7.3.3 RmDir 函数

RmDir()函数的功能是删除一个已经存在的目录，它的语法如下所示：

RmDir(Var S:String)

如果执行以下的程序段：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
ChDir('c:\windows');
//更改工作目录
MkDir('test1');
MkDir('test2');
//在当前的工作目录下创建两个子目录
RmDir('test1');
//在当前的工作目录下删除一个子目录
end;
```

在程序运行的初期，首先通过函数 ChDir()把 c:\windows 设置为当前的工作目录，然后在当前的动作目录下创建了两个子目录 test1 和 test2，最后通过语句 RmDir('test1');，在当前的工作目录下删除一个子目录 test1。

7.3.4 应用实例

下面以一个示例程序来说明 Delphi 中目录类函数的综合应用，在这个程序中，用户可以创建和删除一个目录，实现此功能的具体步骤如下所示：

1. 开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Edit 控件、DriveComboBox 控件、DirectoryListBox 控件和 Button 控件，并且把它们拖放到空白的窗体上，添加控件后的窗体如图 7-15 所示。

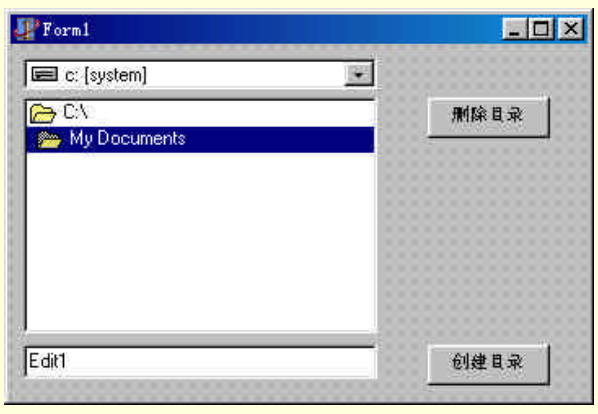


图 7-15 添加控件后的窗体

其中两个按钮的作用是按照文本输入框中的目录名在当前的工作路径下创建和删除一个子目录，而文本输

Green Apple Data Center

入框起到两个方面的作用：

- 一是显示当前选中的目录，即系统工作目录；
- 第二个作用是为创建和删除子目录提供一个子目录名。

2. 设置关联控件

添加控件后，就可以在程序中通过代码设置关联控件。

在程序的设计过程中，用鼠标的左键双击窗体的空白处，就会弹出一个代码窗口，把光标移动到窗体的FormCreate()事件处，添加如下代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.DriveComboBox1.DirList:=form1.DirectoryListBox1;
//设置关联控件
end;
```

程序说明：

当窗体的 FormCreate()事件在窗体的装入的同时就会被激活，进入到窗体的 FormCreate()事件的过程中，通过一条语句设置关联控件。

3. 显示当前路径

在程序的设计过程中，用鼠标的左键双击 DirectoryListBox1 控件，或者用鼠标的左键选择控件 DirectoryListBox1 事件列表框中的 DirectoryListBox1Change()事件，如图 7-16 所示，在屏幕上就会弹出一个显示 DirectoryListBox1Change()事件的代码窗口。

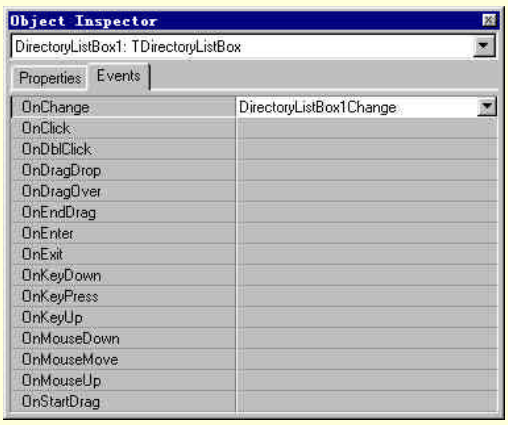


图 7-16 选择控件的 DirectoryListBox1Change()事件

把光标移动到代码窗口中的 DirectoryListBox1Change()的事件处理过程中，并且添加如下代码：

```
procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
form1.Edit1.Text:=form1.DirectoryListBox1.Directory;
//显示当前的路径
end;
```

程序说明：

在程序运行的过程中，当用户改变控件 DirectoryListBox1 的目录时，就会激活控件的 DirectoryListBox1Change()事件，然后程序通过语句 form1.Edit1.Text:=form1. DirectoryListBox1.Directory;就可以显示出系统当前的工作路径。

4. 目录操作

在本程序中，对目录的操作，如创建和删除等动作，都是通过激活两个按钮来实现，所以在程序的设计过程中，用鼠标的左键双击控件“创建目录”按钮（Button1 控件），在代码窗口中就会显示相应的按钮事件，在其中添加如下所示的目录操作代码：

```
procedure TForm1.Button1Click(Sender: TObject);
```

Green Apple Data Center

```
begin
ChDir(form1.DirectoryListBox1.Directory);
//设置当前的路径为系统工作路径
MkDir(form1.Edit1.Text);
//创建一个新的目录
form1.DirectoryListBox1.Update;
//更新目录显示结果
end;
```

程序说明：

当在程序的运行过程中，用鼠标的左键单击按钮时，就会激活响应的按钮事件，虽然创建目录和删除目录的语句略有区别，但是大体的步骤相同，即首先把控件 DirectoryListBox1 **中的当前目录设置为系统当前的工作路径，然后根据控件** Edit1 **中的文本内容来在当前的路径下创建或者删除一个目录，最后通过语句** form1.DirectoryListBox1.Update;**来显示目录更新的结果。**

同样的，在程序的设计阶段用鼠标的左键双击控件 Button2**，在屏幕上就会弹出如图** 7-17 **所示的代码窗口。**

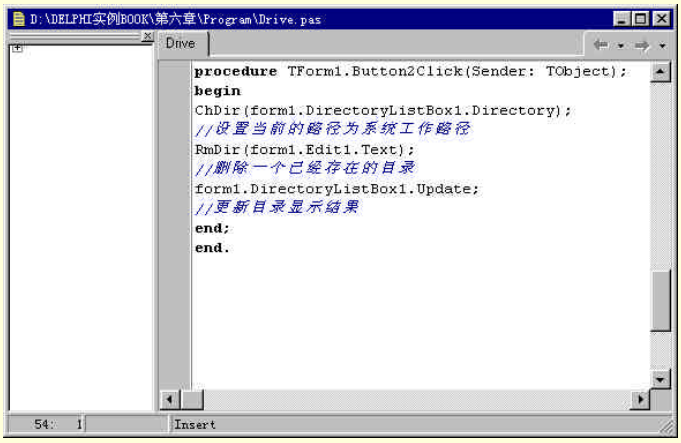


图 7-17　**程序的代码窗口**

将光标移动到控件 Button2Click()**事件的处理过程中，并且添加如下所示的删除指定目录的程序代码：**

```
procedure TForm1.Button2Click(Sender: TObject);
begin
ChDir(form1.DirectoryListBox1.Directory);
//设置当前的路径为系统工作路径
RmDir(form1.Edit1.Text);
//删除一个已经存在的目录
form1.DirectoryListBox1.Update;
//更新目录显示结果
end;
```

5．运行程序

做完以上的工作后，存储文件（这里我们保存项目为 P_directory**，单元文件为** directory**），按键盘上的功能键** F9 **运行程序。**

在程序运行的过程中，将当前系统的工作路径选择为根目录“ e:\ **”，如图** 7-18 **所示，此时** e **盘上只有六个子目录。**

Green Apple Data Center

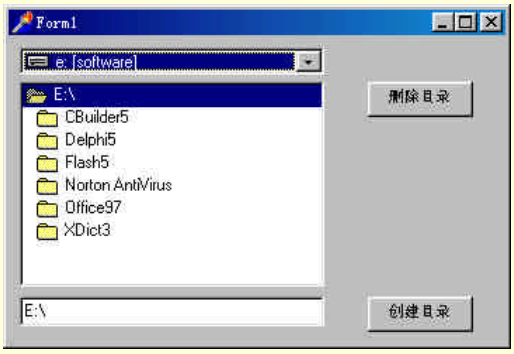


图 7-18　创建了一个“ mkdir ”的新目录

为了创建新目录，我们在 Edit 控件中输入新的目录名为 mkdir，然后用鼠标的左键单击按钮“ 创建目录 ”，程序运行结果如图 7-19 所示，mkdir 目录添加到 e 盘上。

在程序运行过程中，不但可以在系统工作路径下创建一个新的子目录，同时还可以在当前的工作目录下删除一个已经存在的子目录，在文本框中输入欲删除的目录名“ temp ”，然后单击按钮“ 删除目录 ”，即可。

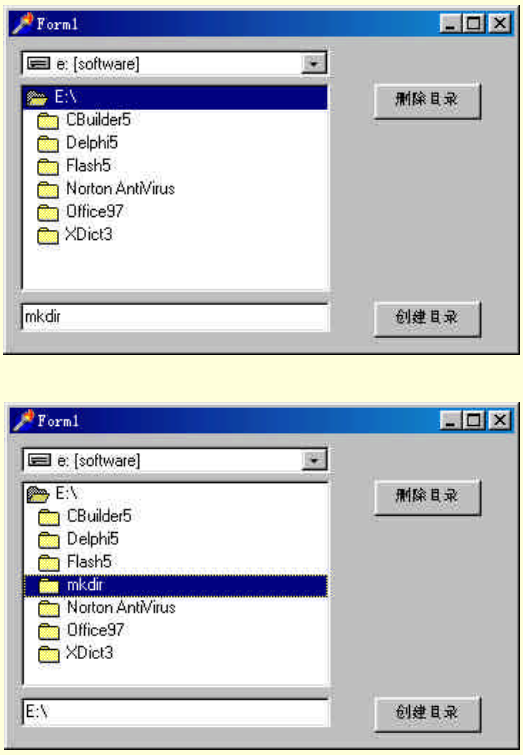


图 7-19　创建新的子目录“ mkdir ”

下面是本应用程序的代码清单，以便读者参考：

程序清单

```
unit directory;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, FileCtrl;
```

Green Apple Data Center

```
type
  TForm1 = class(TForm)
    DriveComboBox1: TDriveComboBox;
    DirectoryListBox1: TDirectoryListBox;
    Button1: TButton;
    Button2: TButton;
    Edit1: TEdit;
    procedure FormCreate(Sender: TObject);
    procedure DirectoryListBox1Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.DriveComboBox1.DirList:=form1.DirectoryListBox1;
//设置关联控件
end;

procedure TForm1.DirectoryListBox1Change(Sender: TObject);
begin
form1.Edit1.Text:=form1.DirectoryListBox1.Directory;
//显示当前的路径
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
ChDir(form1.DirectoryListBox1.Directory);
//设置当前的路径为系统工作路径
MkDir(form1.Edit1.Text);
//创建一个新的目录
form1.DirectoryListBox1.Update;
//更新目录显示结果
end;
```

Green Apple Data Center

```
procedure TForm1.Button2Click(Sender: TObject);
begin
ChDir(form1.DirectoryListBox1.Directory);
//设置当前的路径为系统工作路径
RmDir(form1.Edit1.Text);
//删除一个已经存在的目录
form1.DirectoryListBox1.Update;
//更新目录显示结果
end;
end.
```

7.4 管理驱动器

在 Delphi 的驱动器操作函数中，常用的函数有 DiskSize()、DiskFree()和 GetDir()，它们的功能和使用方法如下。

7.4.1 DiskSize 函数

DiskSize()函数的功能是计算用户指定的驱动器的总容量，即所有的可用磁盘空间，它的语法如下：

DiskSize(Drive:Byte):Longint

其中参数 Drive 的取值为 0 时表示当前驱动器，取 1 时表示 A 驱动器，取 2 时表示 B 驱动器，取 3 时表示 C 驱动器，其它依次类推。

下面的示例就是利用 DiskSize()函数来达到获得计算机上驱动器信息的目的，编程的具体步骤如下所示。

1．开始工作

首先启动一个新的项目，选择菜单“File”中的“New Application”项；

在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 Memo 控件，并且在它的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 Memo 控件，如图 7-20 所示。

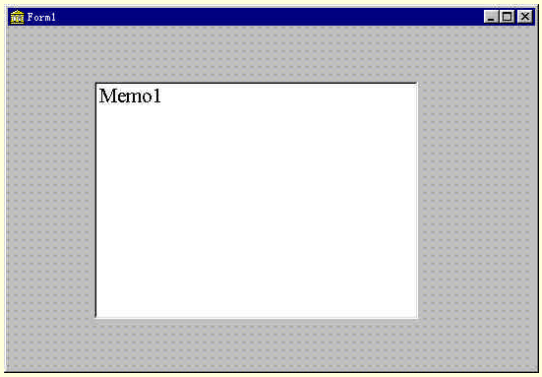


图 7-20 添加控件后的窗体

2．声明变量

本示例程序的作用是获得硬盘上三个驱动器（C、D、E）的可用磁盘空间的信息，所以在程序变量的声明中要定义三个字符串变量 temp1、temp2 和 temp3，其中依次存储 C、D、E 驱动器的可用磁盘空间。

```
var
  Form1: TForm1;
  temp1:string;
  temp2:string;
```

Green Apple Data Center

```
  temp3:string;
```

3. 添加代码

在程序的设计阶段，用鼠标的左键双击窗体上的空白处，在屏幕上就会弹出一个代码窗口，在其中添加如下所示的代码，用于获得和显示驱动器的信息。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
temp1:=inttostr(disksize(3));
temp1:='C 驱动器的可用空间：'+temp1+'Byte'+chr(13)+chr(10);
//获得 C 驱动器信息
temp2:=inttostr(disksize(4));
temp2:='D 驱动器的可用空间：'+temp2+'Byte'+chr(13)+chr(10);
//获得 D 驱动器信息
temp3:=inttostr(disksize(5));
temp3:='E 驱动器的可用空间：'+temp3+'Byte'+chr(13)+chr(10);
//获得 E 驱动器信息
form1.memo1.text:=temp1+temp2+temp3;
//显示驱动器信息
end;
```

4. 运行程序

做完以上的工作后，按键盘上的功能键 F9 运行程序，程序的运行结果如图 7-21 所示。

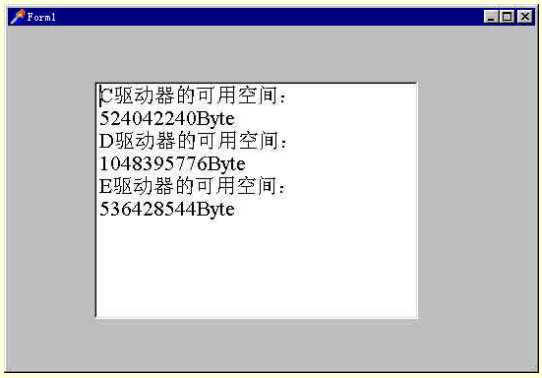


图 7-21　程序运行结果

7.4.2　DiskFree 函数

DiskSize()函数的功能是计算用户指定的驱动器的剩余磁盘空间，它的语法结构如下所示：

DiskFree(Drive:Byte):Longint

其中参数 Drive 的取值为 0 时表示当前驱动器，取 1 时表示 A 驱动器，取 2 时表示 B 驱动器，取 3 时表示 C 驱动器，其它依次类推。

下面的示例就是利用 DiskFree()函数来达到获得计算机上驱动器剩余磁盘空间信息的目的，本示例程序的基本框架是基于 DlskSize()函数的示例程序，但是在其中的代码略有变化，代码如下所示：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
temp1:=inttostr(diskfree(3));
temp1:='C 驱动器的剩余磁盘空间：'+temp1+'Byte'+chr(13)+chr(10);
//获得 C 驱动器信息
temp2:=inttostr(diskfree(4));
temp2:='D 驱动器的剩余磁盘空间：'+temp2+'Byte'+chr(13)+chr(10);
```

Green Apple Data Center

```
//获得 D 驱动器信息
temp3:=inttostr(diskfree(5));
temp3:='E 驱动器的剩余磁盘空间：'+temp3+'Byte'+chr(13)+chr(10);
//获得 E 驱动器信息
form1.memo1.text:=temp1+temp2+temp3;
//显示驱动器信息
end;
```

存储文件，按键盘上的功能键 F9 运行程序，程序运行结果如图 7-22 所示。

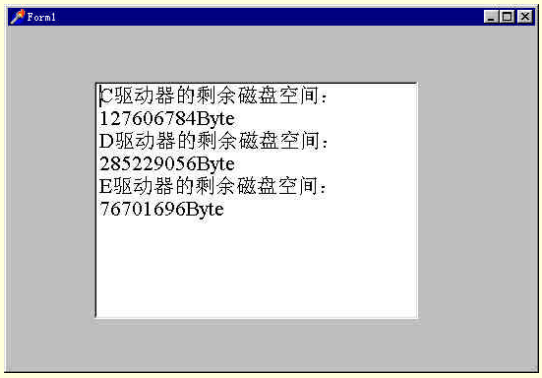


图 7-22　显示驱动器剩余磁盘空间的信息

7.4.3　GetDir 函数

GetDir()函数的功能是返回指定驱动器的当前动作目录，它的语法结构如下所示：

GetDir(Drive：Byte，Var：String)；

其中参数 Drive 的取值为 0 时表示当前驱动器，取 1 时表示 A 驱动器，取 2 时表示 B 驱动器，取 3 时表示 C 驱动器，其它依次类推。

如下所示，当在程序中执行以下的代码时，结果就会在屏幕上输出一个对话框，显示当前系统的工作目录。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
chdir('d:\delphi\bin');
//更改当前的工作目录
getdir(0,s);
//获得工作目录信息
showmessage(s);
//输出当前的工作目录
end;
```

添加代码后，在键盘上按功能键 F9 运行程序，结果如图 7-23 所示，在屏幕上报告当前系统的工作目录为 d:\delphi\bin。



图 7-23　输出信息

Green Apple Data Center

7.5 小 结

文件处理在 Windows 的程序设计中一直是一个比较活跃的领域，同样，在 Delphi 的程序设计中，也把文件处理放在一个比较重要的位置，可以说，文件处理是进行大的应用程序的基础。

在本章中，介绍了在 Delphi 中进行程序设计的几种常用的方法，如通过文件类控件设计文件处理应用程序、调用系统函数来进行系统的目录和文件的维护等，特别为读者准备了两个文件处理应用程序的示例，希望读者在理解的基础上，设计出有个人特色的文件处理应用程序。

Green Apple Data Center

第八章　数据库开发和管理

Delphi 最吸引人之处就是它强大的数据库（DB）功能，利用大量的系统数据库控件（DataSource 控件、Table 控件和 Query 控件等）可以很方便的设计出功能强大的数据库应用程序，不但可以编制访问本地数据库的应用程序，也可以设计能够访问远程大型数据的应用程序。

需要注意的是，本章的重点不是向读者介绍一般的数据库使用原理，而是通过几个具体的应用程序示例来向读者说明在 Delphi 中的数据库的设计方法，所以有很大的实用性。

8.1　使用数据库向导

设计数据库应用程序有两种基本的方法，一是从一个空白的窗体开始开始设计，在窗体上放置各种控件，在代码窗口中添加代码，另一种方法就是利用 Delphi 提供的数据库应用程序向导来完成简单数据库应用程序的设计，下面来介绍一下如何利用向导程序制作数据库应用程序，具体的步骤如下所示。

1．开始工作

首先选择菜单“ File ”/“ New ”选项，在屏幕上就会弹出一个如图 8-1 所示的“ New Items ”对话框。

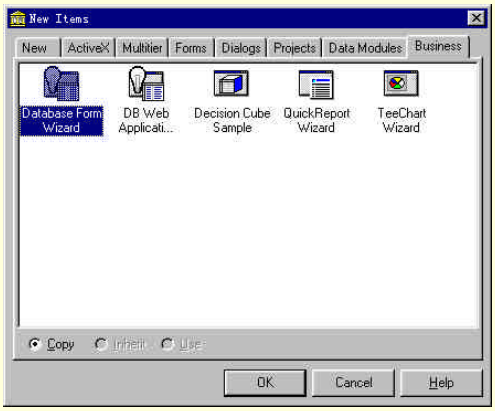


图 8-1　New Items 对话框

在“ New Items ”对话框中选择“ Business ”选项中的“ DatabaseFormWizard ”选项，即打开了一个数据库向导应用程序，单击“ Ok ”按钮进入下一步。

2．选择数据库类型

在如图 8-2 所示的选择数据库类型的对话框中，用户可以对窗体的类型和数据控件类型进行选择，用户可以选择如下类型的窗体：创建一个简单的窗体或创建一个主/细节窗体；同时用户还可以选择如下的数据控件：创建的窗体中包含 Tabel 控件或创建的窗体中包括 Query 控件。

Green Apple Data Center

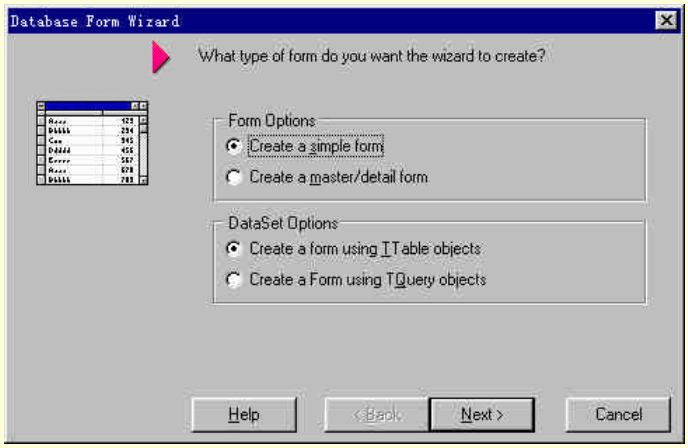


图 8-2　选择数据库类型对话框

选择 Create a simple form 和 Create a form using TTable objects 选项，即创建一个包含 Table 控件的简单窗体。单击“ Next ”按钮进入下一步。

3．选择数据库文件

在如图 8-3 所示的选择数据库文件的对话框中，用户可以选择本地机上的所有有效的数据库文件。

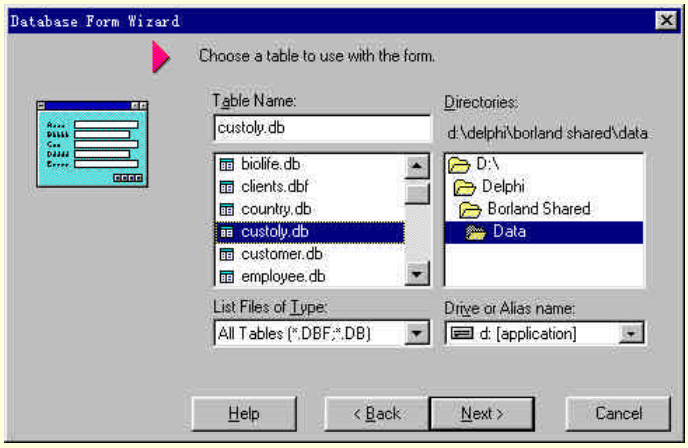


图 8-3　选择数据库文件

在本示例程序中我们选择的数据库文件为“ d:\delphi\borland shared\data\orders.db ”，单击“ Next ”按钮进入下一步。

4．选择显示域

在如图 8-4 所示的对话框中，用户可以选择在程序运行过程中想要显示的域，单击 > 按钮可以向窗体中添加选中的域，而单击 >> 则可以添加所有的域，同样单击 < 按钮和 << 可以删除已经添加的域。

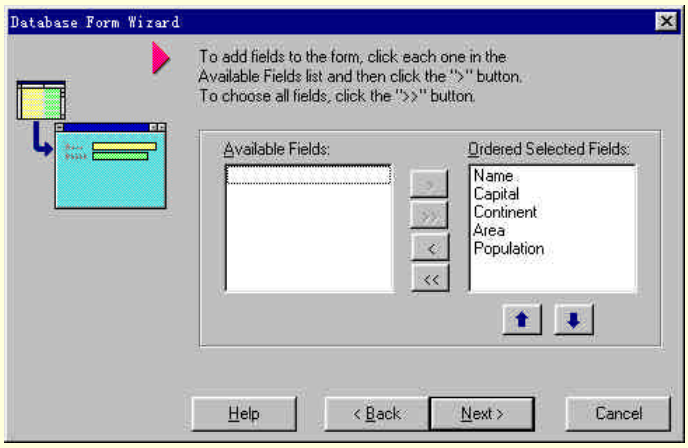


图 8-4　选择显示域

Green Apple Data Center

同时，在选择显示域的对话框中，用户还可以选择各个域在程序运行过程中的索引位置。在本示例程序中，把所有的域都设置为可见，并且，各个域在程序中的索引位置采用默认值，单击“ Next ”按钮进入下一步。

5．选择布局样式

在向导应用程序中，可以选择窗体中的控件的排列方式，如水平排列方式和垂直排列的方式，对话框如图 8-5 所示。

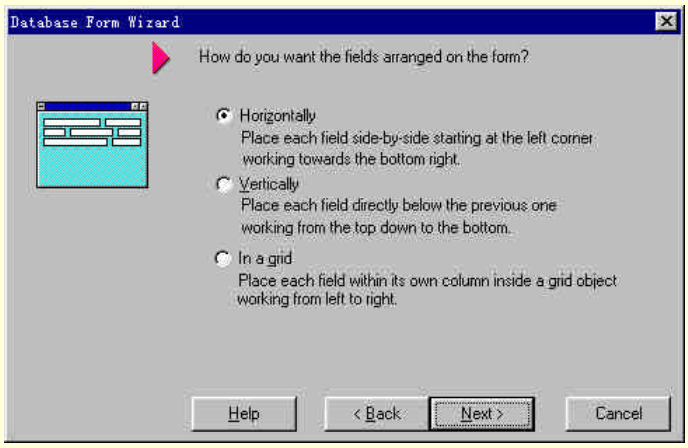


图 8-5　选择布局样式对话框

如果在选择窗体布局样式对话框中选择的是 Horizontally 方式，那么窗体上的控件将会以水平方式对齐，如果选择的是 Vertically 方式，那么窗体上的控件将会以垂直方式排列，而如果选择的是 In a grid，那么在中将会使用 DBGird 控件。

在本示例程序中选择 Horizontally 方式，单击“ Next ”按钮进入下一步。

6．选择窗体形式

在前一步中选择窗体布局后，下一步的工作就是选择窗体的形式，用户可以在如图 8-6 所示的对话框中设置窗体形式的参数。

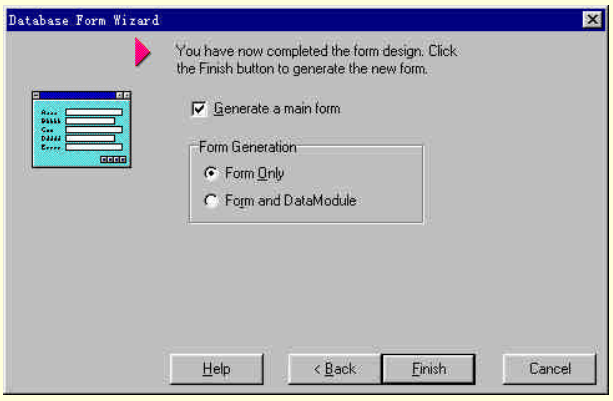


图 8-6　选择窗体形式

用户可以通过复选 Generate a main form 来决定在项目中是否加入一个主窗体。

同时用户还可以选择 Form Only 或者 Form and DataModule，如果选择 Form Only 选项，那么在项目中将只包括一个窗体，所有的控件都会放置在一个窗体中，如果选择 Form and DataModule 选项，那么控件将会分别的放置在两个窗体中。

完成以上的工作后，单击“ Finish ”按钮，就创建了一个简单的数据库应用程序，完成后的窗体如图 8-7 所示。

Green Apple Data Center

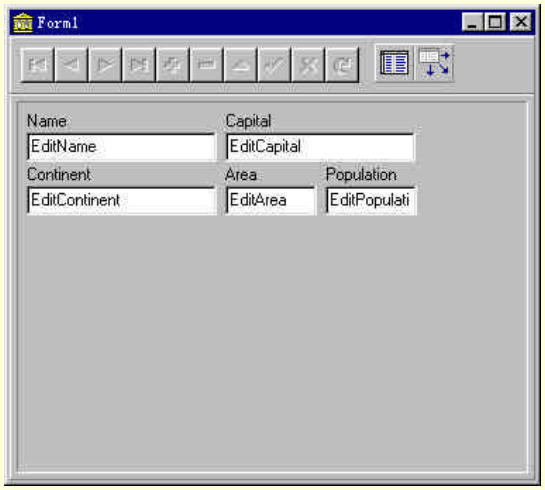


图 8-7　完成后的窗体结构

在窗体上放置有五个 Label **控件、两个** Panel **控件、一个** Table **控件、三个** StringField**、两个** FloatField**、一个** ScrollBox**、五个** DBEdit**、一个** DBNavigator **和一个** DataSource **控件，各个控件的属性设置如下所示：**

```
object Form1: TForm1
Caption = 'Form1'
object Panel1: TPanel
  object DBNavigator: TDBNavigator
    Width = 240
    Height = 25
    DataSource = DataSource1
  end
end
object Panel2: TPanel
  object ScrollBox: TScrollBox
    Width = 361
    Height = 262
    object Label1: TLabel
      Width = 28
      Height = 13
      Caption = 'Name'
    end
    object Label2: TLabel
      Width = 32
      Height = 13
      Caption = 'Capital'
    end
    object Label3: TLabel
      Width = 45
      Height = 13
      Caption = 'Continent'
    end
    object Label4: TLabel
      Width = 22
      Height = 13
```

Green Apple Data Center

```
      Caption = 'Area'
    end
    object Label5: TLabel
      Width = 50
      Height = 13
      Caption = 'Population'
      FocusControl = EditPopulation
    end
    object EditName: TDBEdit
      Width = 135
      Height = 21
      DataField = 'Name'
      DataSource = DataSource1
    end
    object EditCapital: TDBEdit
      Width = 135
      Height = 21
      DataField = 'Capital'
      DataSource = DataSource1
    end
    object EditContinent: TDBEdit
      Width = 135
      Height = 21
      DataField = 'Continent'
      DataSource = DataSource1
    end
    object EditArea: TDBEdit
      Left = 147
      Top = 59
      Width = 65
      Height = 21
      DataField = 'Area'
      DataSource = DataSource1
    end
    object EditPopulation: TDBEdit
      Width = 65
      Height = 21
      DataField = 'Population'
      DataSource = DataSource1
    end
  end
end
object DataSource1: TDataSource
  DataSet = Table1
end
object Table1: TTable
```

Green Apple Data Center

```
    DatabaseName = 'D:\Delphi\Borland Shared\Data'
    TableName = 'country.db'
    object Table1Name: TStringField
      FieldName = 'Name'
    end
    object Table1Capital: TStringField
      FieldName = 'Capital'
    end
    object Table1Continent: TStringField
      FieldName = 'Continent'
    end
    object Table1Area: TFloatField
      FieldName = 'Area'
    end
    object Table1Population: TFloatField
      FieldName = 'Population'
    end
  end
end
```

提示：

当然，用户可以按照符合自己的习惯，将这个窗体的界面修改为中文版式，请读者参看图 8-8。

7．运行程序

完成以上工作后，选择菜单“File”中的“Save All”选项，在弹出的对话框中选择合适的文件名保存文件。然后按键盘上的功能键 F9 运行程序，结果如图 8-8 所示。

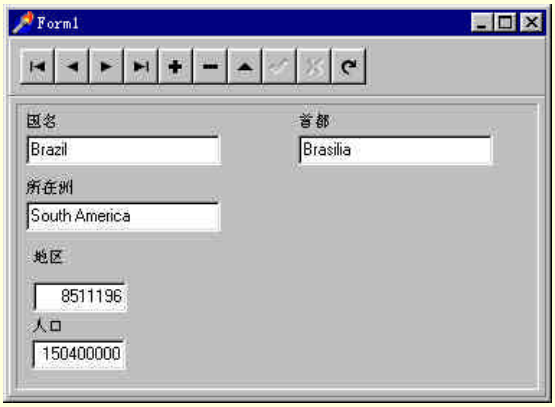


图 8-8　程序运行结果

提示：

本应用程序不需要用户手工添加任何代码，读者只需直接运行程序即可。

8.2　自己定制数据库

在前面通过向导生成的数据库应用程序，所能够实现的功能还是比较简单的，而且数据库的各种操作都是通过 DBNavigator 控件来完成的，下面我们就从一个空白的窗体开始设计一个数据库应用程序。

在本示例程序中，所有的数据库编辑操作都是通过用户自定义的按钮控件来完成的，而且在程序运行的过程中，用户不但可以浏览数据库文件中的各种数据，还可以进行插入、删除等编辑操作，具体的程序设计步骤

Green Apple Data Center

如下所示。

1．开始工作

首先启动一个新的项目，选择菜单“ File ”中的“ New Application ”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择“ Standard ”选项后，在“ Button ”控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个按钮控件，接着向窗体上添加五个 Button 控件、一个 Timer 控件、一个 DBGrid 控件、一个 Table 控件和一个 DataSource 控件，各个控件的功能将在后面分别的加以介绍，添加控件后的窗体如图 8-9 所示。



图 8-9　添加控件后的窗体

其中窗体和控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 252
  Top = 110
  Width = 438
  Height = 369
  Caption = 'Form1'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = True
  PixelsPerInch = 96
  TextHeight = 13
  object DBGrid1: TDBGrid
    Left = 16
    Top = 56
    Width = 401
    Height = 225
    DataSource = DataSource1
    TabOrder = 0
    TitleFont.Charset = DEFAULT_CHARSET
```

Green Apple Data Center

```
    TitleFont.Color = clWindowText
    TitleFont.Height = -11
    TitleFont.Name = 'MS Sans Serif'
    TitleFont.Style = []
  end
  object Button1: TButton
    Left = 24
    Top = 8
    Width = 81
    Height = 33
    Caption = '第一条'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 1
  end
  object Button2: TButton
    Left = 128
    Top = 8
    Width = 81
    Height = 33
    Caption = '上一条'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 2
  end
  object Button3: TButton
    Left = 232
    Top = 8
    Width = 81
    Height = 33
    Caption = '下一条'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 3
```

Green Apple Data Center

```
  end
  object Button4: TButton
    Left = 328
    Top = 8
    Width = 81
    Height = 33
    Caption = '最后一条'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 4
  end
  object Button5: TButton
    Left = 72
    Top = 296
    Width = 81
    Height = 33
    Caption = '添加记录'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 5
  end
  object Button6: TButton
    Left = 256
    Top = 296
    Width = 81
    Height = 33
    Caption = '删除记录'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 6
  end
  object Table1: TTable
    Active = True
    DatabaseName = 'DBDEMOS'
```

Green Apple Data Center

```
    TableName = 'items.db'
    Left = 368
    Top = 192
  end
  object DataSource1: TDataSource
    DataSet = Table1
    Left = 368
    Top = 136
  end
  object Timer1: TTimer
    Interval = 100
    Left = 368
    Top = 88
  end
end
```

在控件的属性设置过程中，一定要注意以下几个控件的属性设置：

- Table **控件：**DatabaseName **属性要设置为**'DBDEMOS'**，这时单击** TableName **属性右侧的下拉式列表框，就可以选择数据库文件**'items.db'**了，最后要把它的** Active **属性设置为** True**；**
- DataSource **控件：**DataSet **属性设置为** Table1**；**
- DBGrid **控件：**DataSource **属性设置为** DataSource1**，即指明了数据显示的来源。**

2．响应按钮事件

在窗体上放置有六个 Button **控件，它们可以实现以下的功能：**

- **控件** Button1**：将数据库的指针移动到第一条记录处；**
- **控件** Button2**：将数据库的指针移动移动到前一条记录处；**
- **控件** Button3**：将数据库的指针移动移动到下一条记录处；**
- **控件** Button4**：将数据库的指针移动到最后一条记录处；**
- **控件** Button5**：在当前位置插入一条数据；**
- **控件** Button6**：将当前位置处的数据删除。**

由于篇幅的关系，在这里仅以 Button1 **控件和** Button5 **控件为例来说明如何添加响应按钮事件的代码。**

在程序设计的过程中，用鼠标的左键双击窗体中的 Button1 **和** Button5 **控件，在屏幕上就会弹出一个代码窗口，把光标分别移动到控件** Button1 **和控件** Button5 **的事件响应过程中，并且添加如下所示的代码：**

```
procedure TForm1.Button1Click(Sender: TObject);
begin
form1.Table1.First;
//移动到第一条记录
end;
……
procedure TForm1.Button5Click(Sender: TObject);
begin
form1.Table1.Insert;
//在当前位置插入一条记录
end;
```

程序中各个按钮功能都是通过调用 Table **控件的相应方法来实现的，如** First **按钮就是通过调用** Table **控件的** First **方法来实现把当前数据库指针移动到第一条记录的功能的，而** Insert **按钮是通过调用** Table **控件的** Insert **方法来实现在当前位置插入一条记录的功能的。**

在程序运行的过程中，当用户用鼠标的左键单击窗体上的各个按钮时，程序就会自动的调用 Table **控件的相**

Green Apple Data Center

应方法来实现对应的功能。

如用户单击 Insert 按钮，程序就会调用 Insert 方法在数据库的当前位置插入一条记录，结果如图 8-10 所示。



图 8-10　在当前位置插入一条记录

其它几个按钮的代码添加过程就不一一的加以介绍了，读者可以参看附后的源程序代码，在这里仅仅列举出各个按钮所对应的方法：

- “第一条”按钮：Table 控件的 First 方法；
- “上一条”按钮：Table 控件的 Prior 方法；
- “下一条”按钮：Table 控件的 Next 方法；
- “最后一条”按钮：Table 控件的 Last 方法；
- “添加记录”按钮：Table 控件的 Insert 方法；
- “删除记录”按钮：Table 控件的 Delete 方法。

3. 相应计时器事件

窗体上添加的 Timer 控件的作用是在程序运行的过程中动态的设置各个按钮控件的有效状态，例如在浏览或者编辑数据库中的数据时，如果当前的数据库指针已经移动到了数据库的开头或结尾，再执行 Tabel 控件的 Last 方法、Next 方法、First 方法和 Prior 方法，程序就会出现运行错误。

为了在程序运行的过程中避免以上错误的出现，在程序中应该有设置各个按钮有效状态的代码，为此，在程序设计的过程中，用鼠标的左键双击窗体上的 Timer 控件，在屏幕上就会弹出一个代码窗口，在其中可以添加动态设置按钮有效状态的代码。

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
if form1.Table1.Eof then
    begin
        form1.Button4.Enabled:=false;
        form1.Button3.Enabled:=false;
    end
else
    begin
        form1.Button4.Enabled:=true;
        form1.Button3.Enabled:=true;
    end;
//动态设置“Next”和“Last”按钮的有效状态
if form1.Table1.Bof then
    begin
        form1.Button1.Enabled:=false;
```

Green Apple Data Center

```
            form1.Button2.Enabled:=false;
        end
    else
        begin
            form1.Button1.Enabled:=true;
            form1.Button2.Enabled:=true;
        end;
    //动态设置“ First ”和“ Prior ”按钮的有效状态
    end;
```

程序说明：

Timer 控件的 Interval 属性设置为 100，即在程序的运行过程中，每隔 100 微秒就会自动的激活一个 Timer1Timer()事件，随时的检测数据库文件指针的位置，同时设置按钮的有效状态。结果如图 8-11 所示。



图 8-11　动态设置按钮有效状态

4. 运行程序

按照附后的源程序，添加剩余的程序代码后，选择菜单“ File ”中的“ Save All ”选项，在弹出的对话框中选择合适的文件名保存文件。然后按键盘上的功能键 F9 运行程序，程序运行的结果如图 8-12 所示。



图 8-12　程序运行结果

附程序完整源代码如下所示：

程序清单

Green Apple Data Center

```
unit DB2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Db, DBTables, Grids, DBGrids, ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Timer1: TTimer;
    DBGrid1: TDBGrid;
    DataSource1: TDataSource;
    Table1: TTable;
    procedure Button1Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
form1.Table1.First;
//移动到第一条记录
end;
```

Green Apple Data Center

```
procedure TForm1.Button5Click(Sender: TObject);
begin
form1.Table1.Insert;
//在当前位置插入一条记录
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
form1.Table1.Prior;
//向前移动一条记录
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
form1.Table1.Next;
//移动到下一条记录
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
form1.Table1.Last;
//移动到最后一条记录
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
form1.Table1.Delete;
//删除当前记录
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
if form1.Table1.Eof then
    begin
          form1.Button4.Enabled:=false;
          form1.Button3.Enabled:=false;
    end
else
    begin
          form1.Button4.Enabled:=true;
        form1.Button3.Enabled:=true;
    end;
//动态设置"Next"和"Last"按钮的有效状态
if form1.Table1.Bof then
    begin
          form1.Button1.Enabled:=false;
```

Green Apple Data Center

```
            form1.Button2.Enabled:=false;
        end
    else
        begin
            form1.Button1.Enabled:=true;
            form1.Button2.Enabled:=true;
        end;
    //动态设置"First"和"Prior"按钮的有效状态
    end;

    procedure TForm1.FormCreate(Sender: TObject);
    begin
    form1.Timer1.Enabled:=false;
    //设置控件的有效状态
    end;
    end.
```

8.3 数据库管理程序

Delphi 提供了两个有关图片数据库的对话框控件——“打开图片”和“保存图片”对话框。利用这两个图片数据库对话框控件，可以很方便的设计出自己的图片数据库管理程序，下面，我们以一个具体的实例来讲解具体的程序设计步骤。

1．建立数据库图表

我们可以用 Delphi 提供的数据库桌面平台，来建立一个小型的数据库表。

在 Delphi 的 Tools 菜单中选择 Database Desktop 命令，接着就会启动 Database Desktop。在此桌面平台上，我们新建一个 Paradox 7 类型的数据库表，表的两个字段分别是“名称”和“图片”，并且将“名称”字段设置为主关键字，如图 8-13 所示。

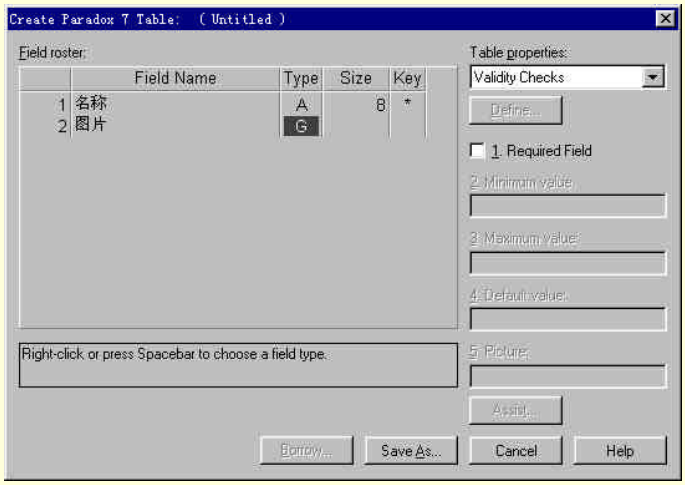


图 8-13 建立图片数据库表

最后单击对话框上的“Save As”按钮，将建立的数据库表保存下来，这里我们保存在 c:\temp 目录，并且命名为 picture.db，接着就直接退出 Database Desktop。

Green Apple Data Center

2．建立窗体

重新返回 Delphi 开发平台，启动一个新的项目，选择菜单“File”中的“New Application”项，在 Delphi 的集成开发环境中就会弹出一个新建的窗体，首先进入工具栏上的“Dialog”选项，为窗体添加一个 OpenPiactureDialog 和一个 SavePictureDialog 控件，然后在控件工具栏上选择“Data control”选项后，在 DBImage 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 DBImage 控件，接着再添加一个 DBEdit 控件，一个 DBNavigator 控件，然后在工具栏上选择“Data Access”选项，再向窗体上添加一个 Table 控件和一个 DataSource 控件，最后向窗体上添加一个标签控件、两个 Button 控件。

各个控件的属性和功能将在后面分别的加以介绍，添加控件后的窗体如图 8-14 所示。



图 8-14　添加控件后的窗体

其中窗体以及窗体上各个控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 419
  Height = 373
  Caption = '数据库'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  OnCreate = FormCreate
  PixelsPerInch = 96
  TextHeight = 13
  object TLabel
    Left = 240
    Top = 168
    Width = 3
    Height = 13
```

Green Apple Data Center

```
  end
  object Label1: TLabel
    Left = 24
    Top = 16
    Width = 40
    Height = 20
    Caption = ' 名字 '
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
  end
  object Button1: TButton
    Left = 32
    Top = 56
    Width = 81
    Height = 33
    Caption = '打开图片'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 0
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 264
    Top = 8
    Width = 81
    Height = 33
    Caption = '存入文件'
    Font.Charset = DEFAULT_CHARSET
    Font.Color = clWindowText
    Font.Height = -16
    Font.Name = 'MS Sans Serif'
    Font.Style = []
    ParentFont = False
    TabOrder = 1
    OnClick = Button2Click
  end
  object DBEdit1: TDBEdit
    Left = 64
```

Green Apple Data Center

```
    Top = 8
    Width = 121
    Height = 25
    DataField = '名称: '
    DataSource = DataSource1
    TabOrder = 2
  end
  object DBNavigator1: TDBNavigator
    Left = 152
    Top = 56
    Width = 250
    Height = 25
    DataSource = DataSource1
    TabOrder = 3
  end
  object DBImage1: TDBImage
    Left = 8
    Top = 96
    Width = 393
    Height = 241
    DataField = '图片'
    DataSource = DataSource1
    TabOrder = 4
  end
  object Table1: TTable
    DatabaseName = 'c:\temp'
    TableName = 'picture.db'
    Left = 360
    Top = 208
  end
  object DataSource1: TDataSource
    DataSet = Table1
    Left = 312
    Top = 184
  end
  object SavePictureDialog1: TSavePictureDialog
    Left = 264
    Top = 144
  end
  object OpenPictureDialog1: TOpenPictureDialog
    Left = 192
    Top = 128
  end
end
```

添加到窗体上的主要控件的作用如下所示：

- DBImage **控件：显示打开的数据库图片；**

Green Apple Data Center

- DBNavigator 控件：实现在程序运行过程中对浏览数据库表的各种操作，如添加记录、删除记录等；
- OpenPictureDialog 控件：打开图片对话框；
- SavePictureDialog 控件：保存图片对话框；
- “打开图片”按钮：实现激活“打开图片”对话框；
- “存入文件”按钮：实现激活“保存图片”对话框。

3. 程序初始化

所谓的程序初始化，在本程序中指的就是在窗体 procedure TForm1.FormCreate(Sender: TObject);事件中添加处理代码。

在程序的设计过程中，首先用鼠标选中窗体，然后在窗体的事件列表中选中 procedure TForm1.FormCreate(Sender: TObject);事件，在屏幕上就会弹出一个代码窗口，把光标移动到事件处理过程中，并且添加如下所示的程序初始化代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
table1.active:=true;
end;
```

程序的初始化代码在程序的运行初期就会被执行，作用是让数据库表处于被打开状态。

然后双击“打开图片”按钮控件，在屏幕上就会弹出相应的事件处理过程，把光标移动到事件处理过趁中，并且添加如下所示的响应代码：

```
procedure TForm1.ToolButton1Click(Sender: TObject);
begin
if openpicturedialog1.execute then
dbimage1.picture.loadfromfile(openpicturedialog1.filename);
end;
```

按照同样的方法，为另外一个按钮添加代码如下：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
if savepicturedialog1.execute then
dbimage1.picture.savetofile(savepicturedialog1.filename);
end;
```

4. 运行程序

添加剩余的程序代码后，选择菜单“File”中的“Save All”选项，在弹出的对话框中选择合适的文件名保存文件。然后按键盘上的功能键 F9 运行程序，程序运行结果如图 8-15 所示。



图 8-15 程序运行结果

Green Apple Data Center

单击数据库导航栏（DBNavigator）上的 + 按钮即可以向数据库表中添加新记录，在“图片名称”栏中输入名称字段，然后单击“打开图片”按钮，这时就会出现“打开图片”对话框，选择一副图片（如图 8-16 所示）。

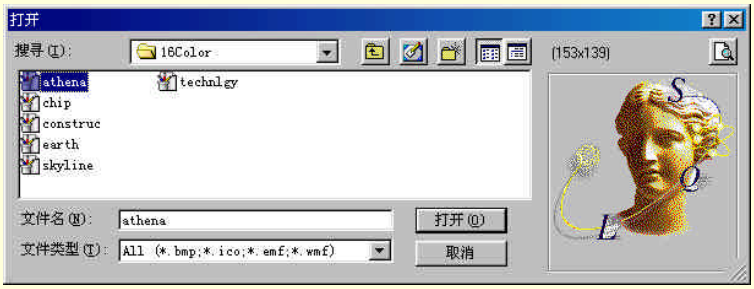


图 8-16　程序运行结果

单击“打开图片”对话框上的“打开”按钮，如图 8-17 所示。然后，单击数据库导航栏上的 ✓ 按钮，本次图片数据库表记录添加就成功了。



图 8-17　数据库添加成功

附程序完整源代码如下所示：

程序清单

```
unit    DBPicture;
interface

uses
   Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
   StdCtrls, ExtDlgs, DBCtrls, ExtCtrls, Mask, Db, DBTables;

type
   TForm1 = class(TForm)
      Table1: TTable;
      DataSource1: TDataSource;
      Button1: TButton;
      Button2: TButton;
```

Green Apple Data Center

```
    DBEdit1: TDBEdit;
    DBNavigator1: TDBNavigator;
    DBImage1: TDBImage;
    SavePictureDialog1: TSavePictureDialog;
    OpenPictureDialog1: TOpenPictureDialog;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
if openpicturedialog1.execute then
dbimage1.picture.loadfromfile(openpicturedialog1.filename);
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
if savepicturedialog1.execute then
dbimage1.picture.savetofile(savepicturedialog1.filename);
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
table1.active:=false;
application.terminate;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
table1.active:=true;
end;
end.
```

Green Apple Data Center

8.4 设计数据库报表

在数据库应用程序设计中要实现的很重要的任务之一就是报表的设计，Delphi 中为用户进行报表设计提供了一个 QuickReport 报表制作工具，利用 QuickReport 可以很简单地制作了打印出报表，用户可以根据实际需要来建立各种各样的报表，而且它的最大的一个优点就是几乎不用程序代码，所有的工作都是在设计的过程中通过设置控件属性来完成的。下面就通过一个具体报表的设计来向读者说明 Delphi 中报表设计的具体步骤和方法。

1. 开始工作

首先启动一个新的项目，选择菜单 File 中的 New 选项项，在 Delphi 的集成开发环境中就会弹出一个如图 8-18 所示的 New Items 对话框。

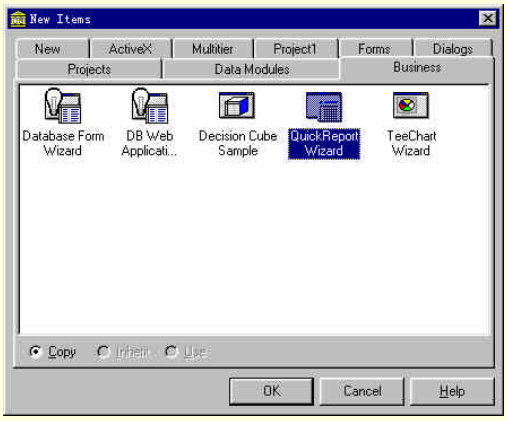


图 8-18 New Items 对话框

在 New Items 对话框中选择 Business 页下面的 QuickReport Wizard 选项，单击 OK 按钮，就开始了报表设计向导，接下来就会显示一个 New Report Wizard 对话框，在其中用鼠标单击 Start Wizard 按钮进入下一步，在弹出的对话框中做如图 8-19 所示的选择。

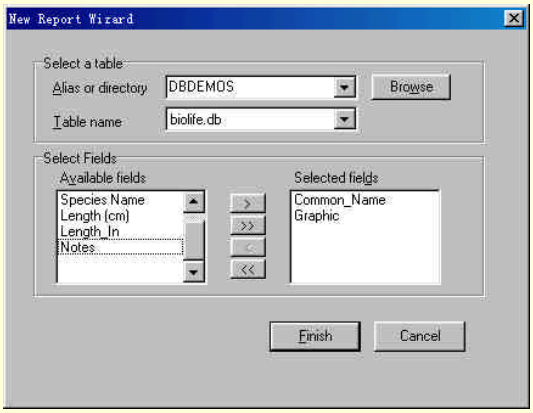


图 8-19 New Report Wizard 对话框

上面一步中，我们在 New Report Wizard 对话框的 Alias or directory 选项框中选择 DBDEMOS，在 Tabel name 选项框中选择 biolife.dbf，在 Selected fields 框中选择 common_name 和 graphic，单击 Finish 按钮就完成了报表的设计。

注意：

- 上面向导设计完成的报表还不能浏览图片，需要进行修改。

2. 修改报表界面

在设计完成的窗体中，在 ColumnHeaderBand 控件上添加一个 QRDBImage 控件 QRDBImage1，QRDBImage 控件的作用是在程序运行的过程中显示数据库中的图像，并且调整 ColumnHeaderBand 和 QRDBImage1 的大小添加控件后的窗体如图 8-20 所示。

Green Apple Data Center

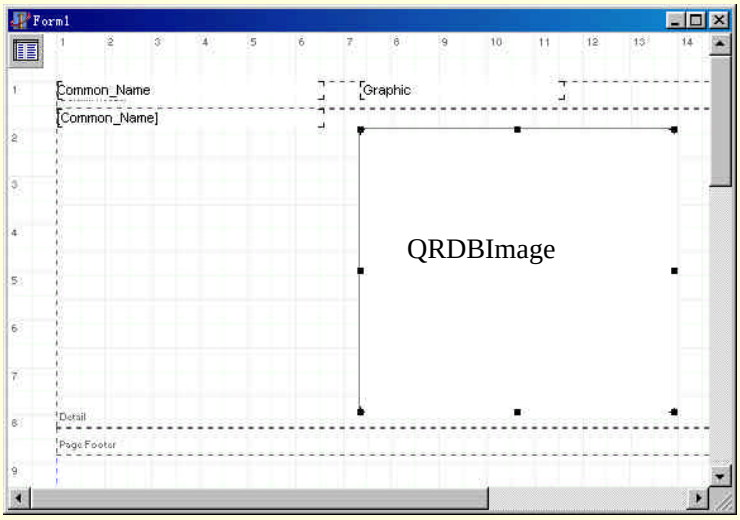


图 8-20　设计完成的窗体

为了实现 QRDBImage1 控件和数据库的接口，请按照如图 8-21 所示对 QRDBImage1 进行属性设置。

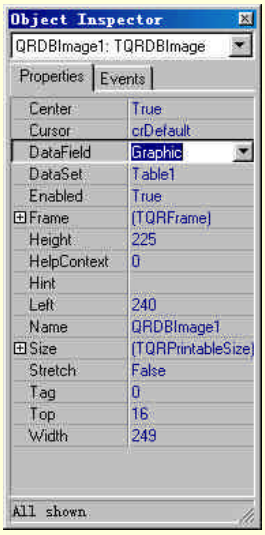


图 8-21　设置 QRDBImage1 控件的属性

本报表窗体用文字描述如下：

object QRDBImage1: TQRDBImage

Left = 392

Top = 0

Width = 129

Height = 121

Frame.Color = clBlack

Frame.DrawTop = False

Frame.DrawBottom = False

Frame.DrawLeft = False

Frame.DrawRight = False

Size.Values = (

320.145833333333

1037.16666666667

0

341.3125)

Green Apple Data Center

```
DataField = 'graphic'
DataSet = Table1
end
```

3. 查看报表

做完以上的工作后，用鼠标选择菜单 File 中的 Save All 选项，在弹出的对话框中选择合适的文件名保存文件。

在 QuickRep 控件上单击鼠标的右键，在窗体上就会弹出一个对话框，在其中选择 Preview 选项，就可以预览设计完成的报表了，结果如图 8-22 所示。

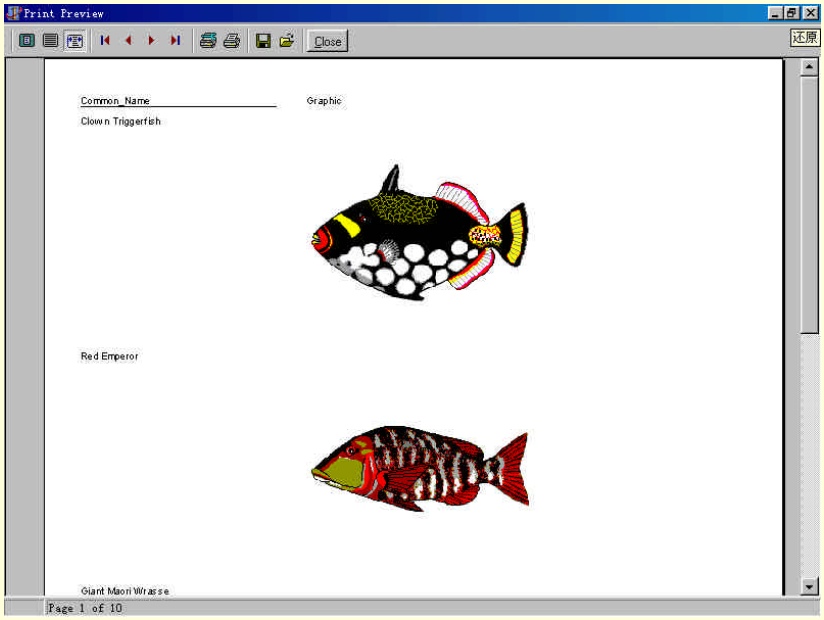


图 8-22 设计完成的报表

8.5 小 结

在本章中，通过了几个典型程序示例向读者展示了 Delphi 的另一特性——强大的数据库开发功能。

在程序的设计过程中，我们利用大量的系统数据库控件（DataSource 控件、Table 控件和 Query 控件等），并且不用添加很多的代码，就可以很方便的设计出功能强大的数据库应用程序。当然，鉴于本书篇幅原因本章讲解的内容有限，希望读者能够参考有关数据库专题开发的书深入学习。

Green Apple Data Center

第九章　多媒体的设计和使用

个人电脑发展到今天，多媒体已经成为 PC 领域不可缺少的一部分，是计算机领域发展最为迅速的市场，而且随着技术的进步，多媒体正在慢慢的成为计算机接口的基础，在市场上也有很多的多媒体设计和制作软件，同 Delphi 相比，或者是功能不够强大，或者是程序设计过于繁琐。

Delphi 5 的出现打破了这种局面，利用 Delphi 5 设计多媒体应用程序是非常简单的，为了方便用户，在 Delphi 5 中提供了一个了多媒体控制接口（MCI）。通过 MCI（多媒体控制接口）这个公用的接口，用户无须介入实际的设备就可以操纵所有的多媒体设备。例如，对 MediaPlayer 控件的简单编程就可以实现视频文件、音频文件的播放和复制等功能。

9.1　认识 MediaPlayer 控件

在 Delphi 5 中，MediaPlayer 控件通常位于控件工具栏上的 System 控件页上，在程序的设计过程中，用户可以通过以下几种方法添加 MediaPlayer 控件。

- 在控件工具栏 System 控件页上用鼠标的左键双击 MediaPlayer 控件的图标，在窗体上就会自动的添加一个 MediaPlayer 控件；
- 在控件工具栏 System 控件页上用鼠标的左键单击击 MediaPlayer 控件的图标，然后把鼠标移动到窗体上，按下的同时移动鼠标，在窗体上就会添加一个 MediaPlayer 控件；

如图 9-1 所示即为添加控件后的窗体。

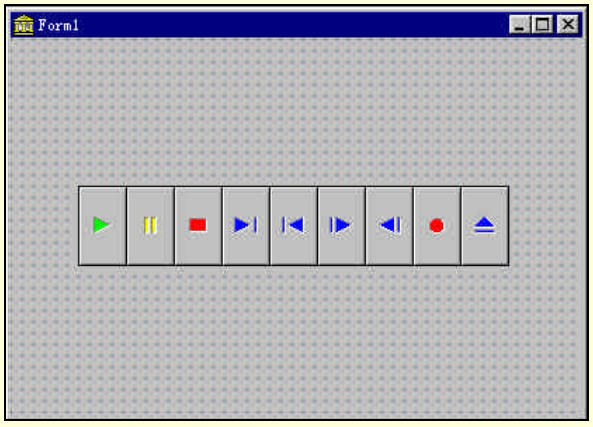


图 9-1　窗体上的 MediaPlayer 控件

MediaPlayer 控件有很多的属性和事件，能够熟练的运用这些属性和事件是进行多媒体程序设计的基础，在本节中，将会对 MediaPlayer 控件中常用的属性和事件作一个简单的介绍，详细的说明请读者参见有关的技术手册。

MediaPlayer 控件中常用的属性如表 9-1 所示。

Green Apple Data Center

表 9-1 MediaPlayer 控件常用的属性

Anchors	AutoEnable	AutoOpen
AutoRewind	ColoredButtons	Constraints
Cursor	DeviceType	Display
Enabled	EnabledButtons	FileName
Height	HelpContext	Hint
Left	Name	ParentShowHint
PopupMenu	Shareable	ShowHint
TabOrder	TabStop	Tag
Top	Visible	VisibleButtons
Width		

下面就对 MediaPlayer 控件中常用的属性作一个简单的说明。

9.1.1 AutoEnable 属性

AutoEnable 属性的功能是设置系统是否具有自动检测 MediaPlayer 控件中各个按钮有效状态能力，它的取值有 True 和 False 两种状态。

如果把 MediaPlayer 控件的 AutoEnable 属性值设置为 True，那么 MediaPlayer 控件就会根据控件和系统当前的状态自动的设置控件中哪个按钮应该处于有效的状态，哪个按钮应该处于无效的状态。

如图 9-2 所示即为控件的 AutoEnable 属性值设置为 True 时程序的运行结果。

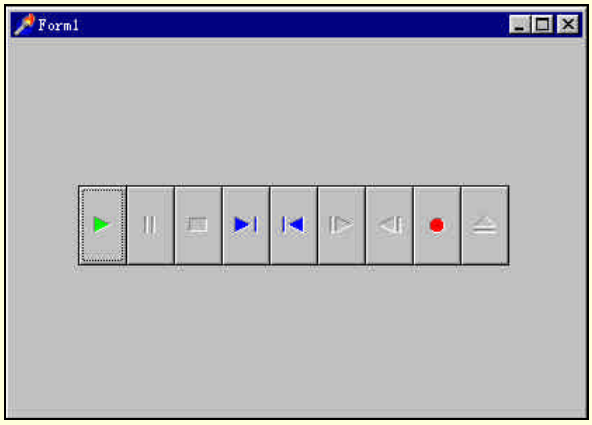


图 9-2 控件的 AutoEnable 属性值设置为 True

在程序的执行过程中，由于已经打开了多媒体播放设备和待播放的声音文件，所以在程序运行的开始阶段，Play 按钮处于有效的状态，而 Pause、Stop 等按钮则处于无效的状态。

实现以上功能的程序代码如下所示，代码的具体功能在后面将详细的加以介绍，在这里就不加赘述了。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.MediaPlayer1.EnabledButtons:=[btPause,btStop,btNext,btPrev,btStep,btBack,btRecord,btEject];
//设置处于有效状态的按钮
form1.MediaPlayer1.FileName:='c:\windows\media\tada.wav';
//设置播放文件名
form1.MediaPlayer1.Open;
//打开多媒体播放设备
form1.MediaPlayer1.AutoEnable:=true;
//设置控件具有自动检测有效状态的功能
end;
```

在程序的运行过程中，单击 Play 按钮，在音箱中就会播放声音文件 tada.wav，在播放的过程中，程序会自动的检测各个按钮的有效状态，如把 Play 按钮设置为无效的状态。

Green Apple Data Center

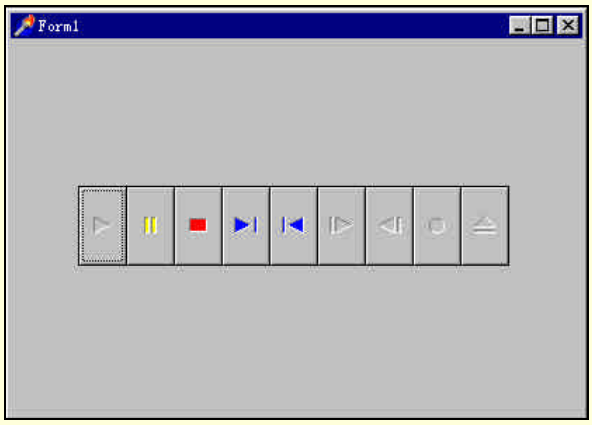


图 9-3　播放文件过程中的按钮有效状态

上面是将控件的 AutoEnable 属性值设置为 True，如果将控件的 AutoEnable 属性值设置为 False 时，MediaPlayer 控件就不会自动的检测和设置控件中各个按钮的有效状态状态,按钮的状态就需要用户自己去设置。

我们把上面的程序代码做一下小改动，如下所示。

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.MediaPlayer1.EnabledButtons:=[btPause,btStop,btNext,btPrev,btStep,btBack,btRecord,btEject];
//设置处于有效状态的按钮
form1.MediaPlayer1.FileName:='c:\windows\media\tada.wav';
//设置播放文件名
form1.MediaPlayer1.Open;
//打开多媒体播放设备
form1.MediaPlayer1.AutoEnable:=false;
//设置控件没有自动检测有效状态的功能
end;
```

做完以上的工作后，存储文件，按键盘上的功能键 F9 运行程序，结果如图 9-4 所示。

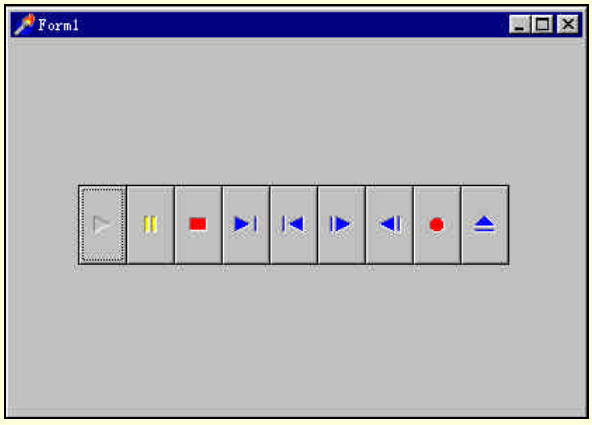


图 9-4　控件的 AutoEnable 属性值设置为 False

从以上的示例可以看出，当把 MediaPlayer 控件的 AutoEnable 属性值设置为 True 时，系统就会屏蔽程序中有关设置控件按钮状态的代码，反之当把 MediaPlayer 控件的 AutoEnable 属性值设置为 False 时，系统不会根据控件的状态来设置按钮的有效状态，这时就需要用户对 MediaPlayer 控件的按钮进行控制。

9.1.2　Display 属性

MediaPlayer 控件的 Display 属性用于设置在多媒体设备播放视频等多媒体文件时的容器，当然，在多媒体程序设计的过程中，也可以不给多媒体播放设备指定播放容器，这样用户就很难控制播放视频的位置，如图 9-5 所示。

Green Apple Data Center



图 9-5　不指定播放容器

在大多数的情况下，在程序中都会给多媒体播放设备指定一个放映的容器，这个容器可以是按钮、文本框，甚至还可以是多媒体播放控件本身，下面就是一个指定按钮为视频播放容器的示例，具体步骤如下所示：

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 5 的集成开发环境中就会弹出一个新建的窗体，在控件工具栏上选择 System 选项后，在 MediaPlayer 控件的图标上双击鼠标的左键，这时空白的窗体上就会出现一个 MediaPlayer 控件，然后在向窗体上添加一个 Panel 控件，其中 Panel 控件用于在程序播放视频的过程中充当容器。添加控件后的窗体如图 9-6 所示。

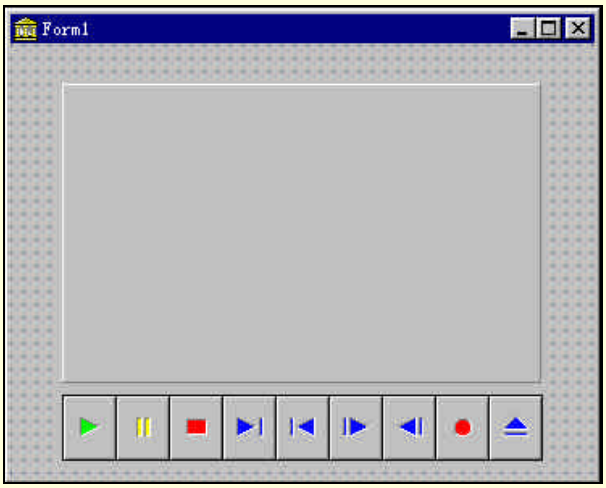


图 9-6　添加控件后的窗体

添加控件后，根据如图 9-6 所示的控件及窗体布局，设置相应的控件和窗体的属性，然后进入下一步。

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，缺省的情况下，光标应该位于在窗体的 FormCreate()事件的过程处理代码段中，在程序的适当位置添加下列代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.MediaPlayer1.AutoEnable:=true;
form1.MediaPlayer1.DeviceType:=dtavivideo;
//设置播放设备类型
form1.MediaPlayer1.FileName:='d:\3dsmaxplugin\bipmos\bipmos\wind.toy\windtoy.avi';
//设置打开的文件名
form1.MediaPlayer1.Open;
//打开多媒体播放设备
form1.MediaPlayer1.display:=form1.Panel1;
//设置播放容器
end;
```

注意：

- 在添加的程序段中，语句 form1.MediaPlayer1.display:=form1.Panel1;它的作用是为程序中播放视频

Green Apple Data Center

指定一个容器，在本程序中指定的视频播放容器为一个 Panel 控件。

做完以上的工作后，选择菜单 File 中的 Save All 选项，在弹出的对话框中选择一个合适的文件名来保存文件。在键盘上按下功能键 F9 运行程序，程序运行的初始画面如图 9-7 所示。

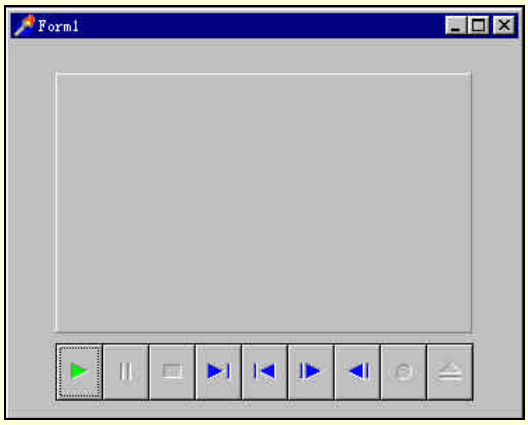


图 9-7 程序运行初始画面

用鼠标的左键单击 Play 按钮，在按钮控件上就会播放指定的视频文件，程序运行结果如图 9-8 所示。

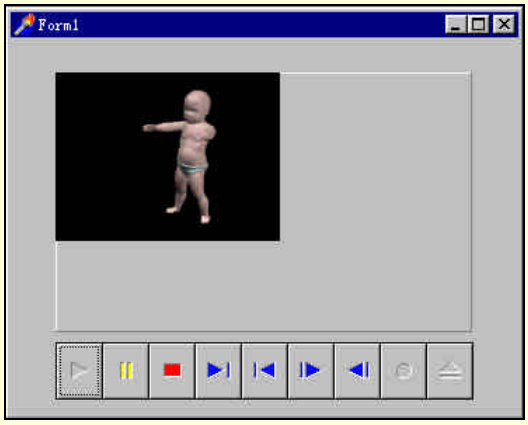


图 9-8 程序运行结果

9.1.3 Filename 属性

MediaPlayer 控件中的 Filename 属性用于为多媒体播放设备指定一个待播放的多媒体文件名，Filename 属性在控件的属性列表框中的位置如图 9-9 所示。

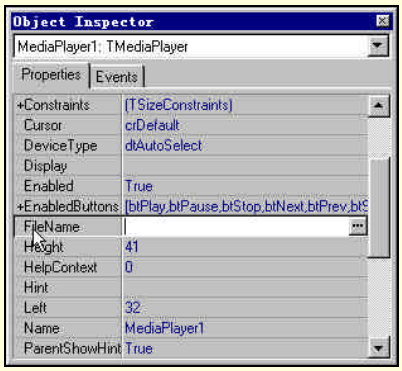


图 9-9 Filename 属性在控件的属性列表框中的位置

在 Filename 属性的设置输入框中可以直接输入多媒体文件的路径和文件名，当然也可以用鼠标左键单击属性输入框右侧的…按钮，就会弹出一个如图 9-10 所示的对话框，在该对话框中用户可以输入或者选择一个有效的多媒体文件，可以打开的多媒体文件类型有 All files (*.*)、 Wave files (*.wav)、Midi files (*.mid)和 Video for Windows (*.avi)等。

Green Apple Data Center

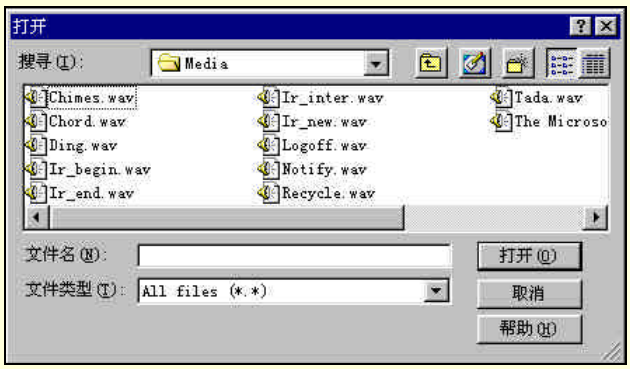


图 9-10 “打开”对话框

9.1.4 DeviceType 属性

MediaPlayer 控件中的 DeviceType 属性用于为打开多媒体设备指定一个设备类型，它的有效取值范围如下：

dtAutoSelect、dtAVIVideo、dtCDAudio、dtDAT、dtDigitalVideo、dtMMMovie、dtOther、dtOverlay、dtScanner、dtSequencer、dtVCR、dtVideodisc 或者 dtWaveAudio

它的类型声明及调用声明如下所示：

```
type TMPDeviceTypes = (dtAutoSelect, dtAVIVideo, dtCDAudio, dtDAT, dtDigitalVideo, dtMMMovie, dtOther, dtOverlay, dtScanner, dtSequencer, dtVCR, dtVideodisc, dtWaveAudio);
property DeviceType: TMPDeviceTypes;
```

MediaPlayer 控件中的 DeviceType 属性的缺省取值为 dtAutoSelect，即程序在运行的过程中会根据所打开的文件类型自己设置设备类型。

用户可以在程序的设计过程中为多媒体设备指定一个设备类型，同样由于 DeviceType 属性在程序的运行过程中是可读可写的，所以可以在程序运行的过程中由程序代码指定，典型的 DeviceType 属性设置语句如下：

```
form1.MediaPlayer1.DeviceType:=dtavivideo;
//设置播放设备类型
```

9.1.5 EnabledButtons 属性

MediaPlayer 控件中的 EnabledButtons 属性用于指定 MediaPlayer 控件中各个按钮的有效状态，而 VisibleButtons 属性则用来设置控件中的各个按钮的可见状态，在缺省的状态下，这两个控件的设置值为：[btPlay,btPause,btStop,btNext,btPrev,btStep,btBack,btRecord,btEject]，即所有的按钮都处于有效和可见的状态，如图 9-11 为 EnabledButtons 属性和 VisibleButtons 属性都设置为缺省值时的显示情况。

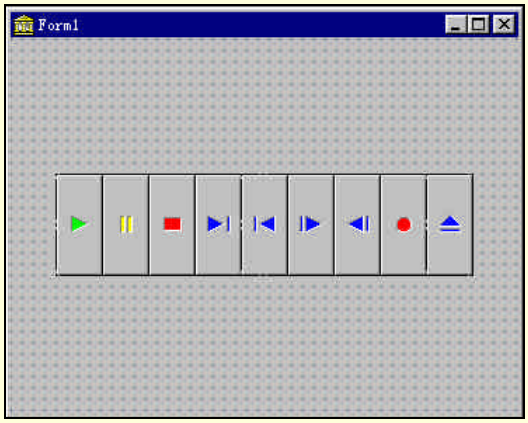


图 9-11 添加控件后的窗体

如果在程序的设计过程中，用鼠标的左键双击窗体的空白处，在弹出的代码窗口中添加下列代码：

```
procedure TForm1.FormCreate(Sender: TObject);
begin
```

Green Apple Data Center

```
form1.MediaPlayer1.EnabledButtons:=[btPlay,btPause,btStop,btNext,btPrev,btStep,btBack,btRecord,btEject];
//设置控件中各个按钮的有效状态
form1.MediaPlayer1.VisibleButtons:=[btPlay,btPause,btStop,btNext,btPrev,btStep,btBack,btRecord, btEject]；
//设置控件中各个按钮的可见状态
end;
```

完成以上工作后，按键盘上的功能键 F9 运行程序，结果如图 9-12 所示。

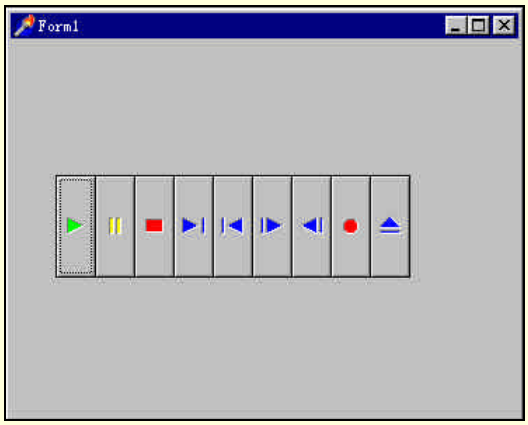


图 9-12　程序运行结果

当执行程序时，首先装入窗体，然后执行窗体 FormCreate()事件中的代码，即通过语句

```
form1.MediaPlayer1.EnabledButtons:=[btPlay,btPause,btStop,btNext,btPrev,btStep,btBack,btRecord,btEject];
```

来设置控件中各个按钮的有效状态；
通过语句

```
form1.MediaPlayer1.VisibleButtons:=[btPlay,btPause,btStop,btNext,btPrev,btStep,btBack,btRecord,btEject]；
```

来设置控件中各个按钮的可见状态。

9.2　音频播放程序

以上我们分别的讲述了 MediaPlayer 控件常用的属性和方法，下面将利用这些属性和方法编制一些多媒体处理应用程序。

我们将利用前面讲到的 MediaPlayer 控件常用的属性和方法，来制作一个 MIDI 音乐播放器程序，它的功能是能够任意的选择一个有效的以*.mid 为后缀的音乐文件，然后在程序中播放，在播放音乐的同时，在窗体中还将有一个滚动条来随时的显示声音文件的播放进度，设计步骤如下：

1．开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 5 的集成开发环境中就会弹出一个新建的窗体，然后向空白的窗体添加四个 Button 控件、一个 ScrollBar 控件、一个 MediaPlayer 控件、一个 OpenDialog 控件和一个 Timer 控件，添加控件后的窗体如图 9-13 所示。

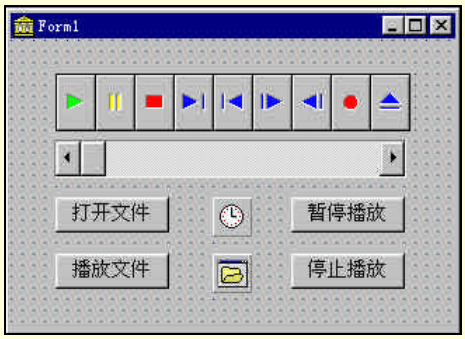


图 9-13　添加控件后的窗体

Green Apple Data Center

所添加的控件的功能如下所示：

- MediaPlayer 控件：为播放音乐文件和系统多媒体设备之间建立连接的关系，同时负责打开多媒体播放设备；
- Button 控件：实现打开文件、播放文件、暂停播放和停止播放等功能；
- ScrollBar 控件：显示程序中播放声音文件的进度，并且能够通过拖动滚动条来手动控制播放的进度；
- OpenDialog 控件：它的作用是显示一个的“打开”的对话框，同时还能够实现打开音乐文件的功能，在其中还可以对多媒体文件进行筛选操作；
- Timer 控件：为程序运行提供时间控制，如随时显示声音文件播放进度。

2．属性设置

添加控件后的下一步工作就是为所添加的控件设置各自的属性，窗体与控件的属性设置可以在程序的设计阶段通过改变属性列表框中的内容来达到，同样，也可以在程序中通过代码来达到，这两种方法的效果是相同的。

在本示例程序中，窗体及控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 325
  Height = 233
  Caption = 'Form1'
  OnClose = FormClose
  OnCreate = FormCreate
  object MediaPlayer1: TMediaPlayer
    Left = 32
    Top = 24
    Width = 253
    Height = 41
  end
  object ScrollBar1: TScrollBar
    Left = 32
    Top = 72
    Width = 249
    Height = 25
    LargeChange = 5
    OnScroll = ScrollBar1Scroll
  end
  object Button1: TButton
    Left = 32
    Top = 112
    Width = 81
    Height = 25
    Caption = '打开文件'
    OnClick = Button1Click
  end
  object Button2: TButton
    Left = 32
    Top = 152
```

Green Apple Data Center

```
    Width = 81
    Height = 25
    Caption = '播放文件'
    OnClick = Button2Click
  end
  object Button3: TButton
    Left = 200
    Top = 112
    Width = 81
    Height = 25
    Caption = '暂停播放'
    OnClick = Button3Click
  end
  object Button4: TButton
    Left = 200
    Top = 152
    Width = 81
    Height = 25
    Caption = '停止播放'
    OnClick = Button4Click
  end
  object Timer1: TTimer
    Interval = 100
    OnTimer = Timer1Timer
    Left = 144
    Top = 112
  end
  object OpenDialog1: TOpenDialog
    Filter = 'MIDI 音乐文件(*.mid)|*.mid'
    Left = 144
    Top = 152
  end
end
```

3. 程序的初始化

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到窗体的 FormCreate()**事件处理代码中，并且添加如下所示的程序初始化代码：**

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Timer1.Enabled:=false;
//设置计时器工作状态
form1.Button2.Enabled:=false;
form1.Button3.Enabled:=false;
form1.Button4.Enabled:=false;
//设置按钮有效状态
end;
```

窗体 FormCreate **事件中的代码在程序运行初期就会被执行，由于在程序刚刚开始运行时还没有选择有效的**

Green Apple Data Center

音乐文件，所以 Timer1 控件的有效状态设置为无效，然后通过三条语句 form1.Button2.Enabled:=false;、form1.Button3.Enabled:=false;和 form1.Button4.Enabled:=false;设置了按钮控件的有效状态。

经过以上初始化处理后的窗体如图 9-14 所示。



图 9-14 程序初始化后的窗体

4. 打开一个音乐文件

在“打开文件”的按钮上双击鼠标的左键，就会弹出一个代码窗口，在其中就可以添加对按钮“打开文件”的响应代码如下所示。

```
procedure TForm1.Button1Click(Sender: TObject);
begin
form1.OpenDialog1.InitialDir:='e:\midi';
//设置对话框缺省路径
form1.OpenDialog1.Title:='请选择一个音乐文件名:';
//设置对话框标题
form1.OpenDialog1.DefaultExt:='mid';
//设置对话框缺省文件名
if form1.OpenDialog1.Execute then
    begin
    form1.MediaPlayer1.FileName:=form1.OpenDialog1.FileName;
    //返回音乐文件名
    form1.MediaPlayer1.Open;
    //打开多媒体播放设备
    form1.Timer1.Enabled:=true;
    form1.Button2.Enabled:=true;
    //设置按钮有效状态
    end;
end;
```

在程序运行的过程中，如果用户用鼠标的左键单击按钮“打开文件”，就会激活控件的 Button1Click()事件，程序首先设置对话框缺省路径为 e:\midi，对话框的标题是“请选择一个音乐文件名:”，在对话框中显示的文件缺省扩展名是 mid，然后调用 OpenDialog1 控件的 Execute 方法来显示一个对话框，在其中用户可以选择一个有效的音乐文件，最后通过 form1.MediaPlayer1.Open;来打开一个多媒体播放设备，并且设置了按钮的有效状态。

5. 播放音乐文件

在窗体上的按钮“播放文件”的作用是播放一个用户选中的音乐文件，在 MediaPlayer 控件中有一个 Play 方法可以在程序运行的过程中播放一个打开的音乐文件，为此，在按钮“播放文件”的响应过程中添加如下代码：

```
procedure TForm1.Button2Click(Sender: TObject);
begin
```

Green Apple Data Center

```
form1.MediaPlayer1.Play;
//播放音乐
form1.Button3.Enabled:=true;
form1.Button4.Enabled:=true;
//设置按钮有效状态
end;
```

值得注意的是，在程序的运行过程中，只有用户在对话框中选择一个有效的音乐文件，单击“打开”按钮，才能成功的向系统多媒体设备传送文件的路径和将要打开的文件名，这时单击“播放文件”按钮才能调用 MediaPlayer 控件中的 Play 方法来播放一个选中的音乐文件。

6．响应计时器事件

窗体上 Timer 控件的作用是为程序运行提供时间控制，如随时显示音乐文件播放进度等。在本示例程序中，Timer 控件的时间控制设置为 100，它的意思是在程序运行的过程中，每隔 100 毫秒就会自动的激活一个 Timer1Timer()事件，为了在程序中动态的显示音乐文件播放的进度，可以在 Timer1Timer()事件中添加如下代码：

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
form1.ScrollBar1.Min:=0;
form1.ScrollBar1.Max:=form1.MediaPlayer1.Length;
//设置滚动条参数
form1.ScrollBar1.Position:=form1.MediaPlayer1.Position;
//随时跟踪滚动条位置
end;
```

在程序运行的过程中，如果控件 Timer1 处于有效的状态，那么每隔 100 毫秒就会自动的激活一个 Timer1Timer()事件，通过语句 form1.ScrollBar1.Position:=form1.MediaPlayer1.Position;在程序中动态的显示音乐文件播放的进度。

7．调节音乐播放进度

在本示例程序中，对音乐播放进度的调节是通过滚动条来实现，响应滚动条动态拖动的事件是 ScrollBar1Scroll，为此，添加如下所示的代码：

```
procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
   var ScrollPos: Integer);
begin
form1.MediaPlayer1.Pause;
//暂停播放
form1.MediaPlayer1.Position:=form1.ScrollBar1.Position;
//拖放播放进度
form1.MediaPlayer1.Play;
//开始播放
end;
```

如果在程序运行的过程中用户用鼠标左键拖动滚动条上的滑动块，就会激活控件的 ScrollBar1Scroll 事件，程序首先调用 MediaPlayer1 控件的 Pause 方法来暂停音乐的播放，然后把当前音乐播放的进度通过语句 form1.MediaPlayer1.Position:=form1.ScrollBar1.Position;调节到当前滑动块的位置，最后调用 MediaPlayer1 控件的 Play 方法重新播放音乐文件。

8．响应窗体的关闭事件

在进行多媒体程序设计的过程中，一定要注意这样的问题，就是在结束程序运行之前正常的关闭多媒体设备，虽然这样做对程序本身的运行不会带来太大的影响，但是养成一个良好的编程习惯对读者以后的工作肯定大有好处。

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
```

Green Apple Data Center

```
begin
form1.MediaPlayer1.Close;
//关闭多媒体播放设备
end;
```

在程序结束运行时，首先会关闭窗体，通过代码 form1.MediaPlayer1.Close;**来关闭多媒体播放设备，这样就不会发生多个应用程序之间抢占播放设备的问题了。**

9．运行程序

在这里对程序其它代码的添加过程就不一一加以介绍了，如果读者有什么不清楚之处，请参看附后的源程序代码。做完以上的工作后，存储文件，按键盘上的功能键 F9 **运行程序。**

程序运行结果如图 9-15 **所示。**

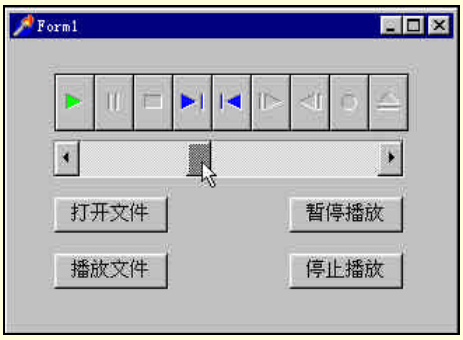


图 9-15　**程序运行结果**

在程序运行的过程中，由于没有给多媒体播放设备指定程序播放文件，所以窗体上的“播放文件”、“暂停播放”和“停止播放”等按钮都处于无效的状态。用鼠标的左键单击“打开文件”按钮，在屏幕上就会弹出一个提示用户选择音乐文件的对话框。

在对话框中用键盘输入一个路径和文件名，或者用鼠标来选择一个有效的音乐文件，单击“打开”按钮返回到程序运行的画面中，这时窗体上的“播放文件”按钮变为有效的状态，单击“播放文件”按钮，在音箱中就会播放选中的音乐文件，同时滚动条会动态的显示出音乐播放的进度。

示例程序中的完整代码如下所示。

程序清单

```
unit MIDI;
interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  MPlayer, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    MediaPlayer1: TMediaPlayer;
    ScrollBar1: TScrollBar;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Timer1: TTimer;
    OpenDialog1: TOpenDialog;
```

Green Apple Data Center

```
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
      var ScrollPos: Integer);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);

var
  Form1: TForm1;
implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.Timer1.Enabled:=false;
//设置计时器工作状态
form1.Button2.Enabled:=false;
form1.Button3.Enabled:=false;
form1.Button4.Enabled:=false;
//设置按钮有效状态
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
form1.OpenDialog1.InitialDir:='e:\midi';
//设置对话框缺省路径
form1.OpenDialog1.Title:='请选择一个音乐文件名:';
//设置对话框标题
form1.OpenDialog1.DefaultExt:='mid';
//设置对话框缺省文件名
if form1.OpenDialog1.Execute then
    begin
    form1.MediaPlayer1.FileName:=form1.OpenDialog1.FileName;
    //返回音乐文件名
    form1.MediaPlayer1.Open;
    //打开多媒体播放设备
    form1.Timer1.Enabled:=true;
    form1.Button2.Enabled:=true;
    //设置按钮有效状态
    end;
end;

procedure TForm1.Button2Click(Sender: TObject);
```

Green Apple Data Center

```
begin
form1.MediaPlayer1.Play;
//播放音乐
form1.Button3.Enabled:=true;
form1.Button4.Enabled:=true;
//设置按钮有效状态
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
form1.MediaPlayer1.Pause;
//暂停播放或继续播放
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
form1.MediaPlayer1.Stop;
//停止播放音乐
form1.Button3.Enabled:=false;
form1.Button4.Enabled:=false;
//设置按钮有效状态
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
form1.ScrollBar1.Min:=0;
form1.ScrollBar1.Max:=form1.MediaPlayer1.Length;
//设置滚动条参数
form1.ScrollBar1.Position:=form1.MediaPlayer1.Position;
//随时跟踪滚动条位置
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
   var ScrollPos: Integer);
begin
form1.MediaPlayer1.Pause;
//暂停播放
form1.MediaPlayer1.Position:=form1.ScrollBar1.Position;
//拖放播放进度
form1.MediaPlayer1.Play;
//开始播放
end;

procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
form1.MediaPlayer1.Close;
```

Green Apple Data Center

```
//关闭多媒体设备
end;
end.
```

9.3 视频播放程序

前面详细的讲述了如何利用 MediaPlayer 控件制作声音播放程序的示例，但是作为多媒体本身来讲，不仅仅包括声音，还应该有视频，所以接下来制作一个能够播放视频文件的视频播放程序。

在 Windows 98 和 Windows 2000 的操作系统中，有一个叫做“ ActiveMoive 控制 ”的应用程序，它不但可以播放以*.avi 为结尾的视频文件，同时也可以播放其它类型的视频文件，甚至可以播放声音文件，利用“ ActiveMoive 控制 ”应用程序来播放以*.avi 为后缀的视频文件的显示情况如图 9-16 所示。

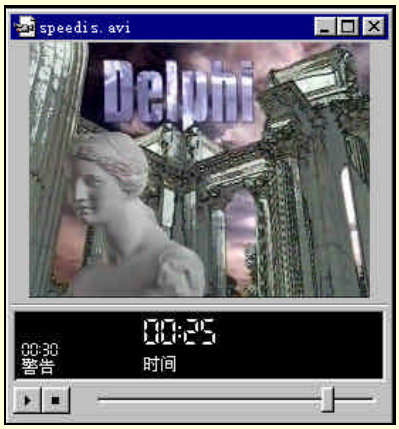


图 9-16 视频播放窗口

下面我们就自己动手来制作一个视频播放器，预计实现的功能如下：

- 能够从文件列表中选择想要播放的文件；
- 在程序中 MediaPlayer 控件处于不可见状态；
- 视频的播放等操作是通过用户自定义的按钮完成的；
- 在程序中能够通过一个滚动条来显示视频的播放进度。

能够实现以上功能的视频播放器程序制作步骤如下所示。

1．开始工作

首先启动一个新的项目，选择菜单 File 中的 New Application 项，在 Delphi 5 的集成开发环境中就会弹出一个新建的窗体，然后向空白的窗体添加八个 Button 控件、一个 ScrollBar 控件、一个 MediaPlayer 控件、一个 OpenDialog 控件和一个 Timer 控件，添加控件后的窗体如图 9-17 所示。

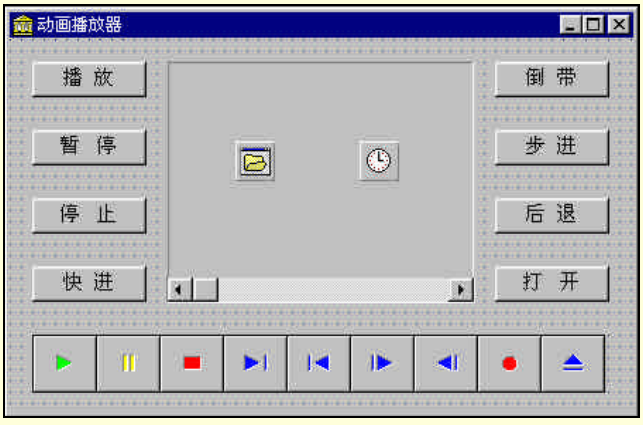


图 9-17 添加控件后的窗体

Green Apple Data Center

所添加的控件的功能如下所示：

- MediaPlayer 控件：为播放视频文件和系统多媒体设备之间建立连接的关系，同时负责打开多媒体播放设备；
- Button 控件：实现打开文件、播放文件和步进播放视频等功能；
- ScrollBar 控件：显示程序中播放视频的进度，同时还可以手动调节视频播放进度；
- OpenDialog 控件：它的作用是显示一个提示用户选择视频文件的对话框，同时还能够实现打开视频文件的功能，在其中还可以对多媒体文件进行筛选操作；
- Timer 控件：为程序运行提供时间控制，如随时显示视频文件播放进度。

2．属性设置

添加控件后的下一步工作就是为所添加的控件设置各自的属性，在本示例程序中，窗体及控件的属性设置如下所示：

```
object Form1: TForm1
  Left = 192
  Top = 107
  Width = 450
  Height = 290
  Caption = '视频播放器'
  OnClose = FormClose
  OnCreate = FormCreate
  object MediaPlayer1: TMediaPlayer
    Left = 16
    Top = 208
    Width = 406
    Height = 41
    DeviceType = dtAVIVideo
    Display = Panel1
  end
  object Button1: TButton
    Left = 16
    Top = 16
    Width = 81
    Height = 25
    Caption = '播　放'
    OnClick = Button1Click
  end
  object Panel1: TPanel
    Left = 112
    Top = 16
    Width = 217
    Height = 153
  end
  object Button3: TButton
    Left = 16
    Top = 112
    Width = 81
    Height = 25
```

Green Apple Data Center

```
    Caption = '停　　止'
    OnClick = Button3Click
  end
  object Button4: TButton
    Left = 16
    Top = 160
    Width = 81
    Height = 25
    Caption = '快　进'
    OnClick = Button4Click
  end
  object Button5: TButton
    Left = 344
    Top = 16
    Width = 81
    Height = 25
    Caption = '倒　带'
    OnClick = Button5Click
  end
  object Button6: TButton
    Left = 344
    Top = 64
    Width = 81
    Height = 25
    Caption = '步　进'
    OnClick = Button6Click
  end
  object Button7: TButton
    Left = 344
    Top = 112
    Width = 81
    Height = 25
    Caption = '后　退'
    OnClick = Button7Click
  end
  object Button8: TButton
    Left = 344
    Top = 160
    Width = 81
    Height = 25
    Caption = '打　　开'
    OnClick = Button8Click
  end
  object Button2: TButton
    Left = 16
    Top = 64
```

Green Apple Data Center

```
    Width = 81
    Height = 25
    Caption = '暂　　停'
    OnClick = Button2Click
  end
  object ScrollBar1: TScrollBar
    Left = 112
    Top = 168
    Width = 217
    Height = 17
    OnScroll = ScrollBar1Scroll
  end
  object OpenDialog1: TOpenDialog
    Left = 160
    Top = 72
  end
  object Timer1: TTimer
    Interval = 100
    OnTimer = Timer1Timer
    Left = 248
    Top = 72
  end
end
```

3．程序的初始化

在程序的设计阶段，用鼠标的左键双击窗体的空白处，在屏幕上就会弹出一个代码窗口，把光标移动到窗体的 FormCreate()**事件处理代码中，并且添加如下所示的程序初始化代码：**

```
procedure TForm1.FormCreate(Sender: TObject);
begin
form1.MediaPlayer1.Visible:=True;
form1.MediaPlayer1.Display:=form1.Panel1;
//设置播放容器
form1.Button1.Enabled:=false;
form1.Button2.Enabled:=false;
form1.Button3.Enabled:=false;
form1.Button4.Enabled:=false;
form1.Button5.Enabled:=false;
form1.Button6.Enabled:=false;
form1.Button9.Enabled:=false;
form1.Button8.Enabled:=true;
//设置按钮有效状态
form1.ScrollBar1.Enabled:=false;
form1.Timer1.Enabled:=false;
//设置控件有效状态
end;
```

在开始执行程序时，系统首先激活窗体的 FormCreate()**事件，然后通过代码** form1.Media Player1.Display:=form1.Panel1;**设置视频播放的容器是窗体上的** Panel1 **控件，最后设置了窗体上各个控件的有效**

Green Apple Data Center

状态和可见状态。

经过以上初始化处理后的窗体如图 9-18 所示。

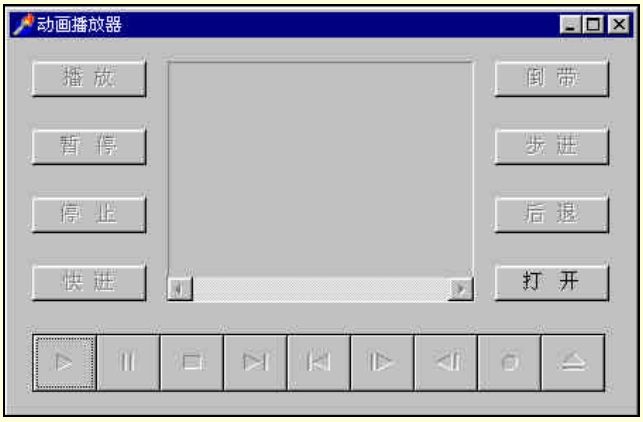


图 9-18　初始化后的窗体

4. 响应计时器事件

窗体上 Timer 控件的作用是为程序运行提供时间控制，如随时显示视频文件播放进度等，为此就需要在程序的运行阶段将 Timer 控件设置为有效的状态，并且为控件指定一个响应的时间长度。

在本示例程序中，Timer 控件的时间控制设置为 100，它的意思是在程序运行的过程中，每隔 100 毫秒就会自动的激活一个 Timer1Timer()事件，为了在程序中动态的显示视频文件播放的进度，可以在程序设计的过程中用鼠标双击窗体中的 Timer 控件，在屏幕上就会弹出一个如图 9-19 所示的代码窗口。

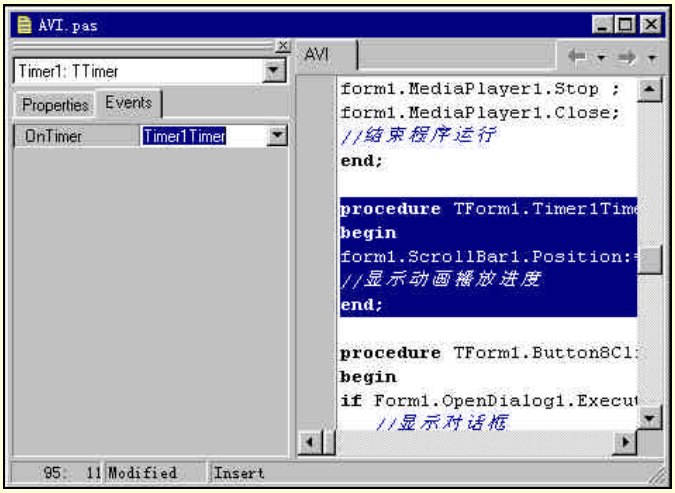


图 9-19　Timer 控件的代码窗口

将光标移动到代码窗口中的 Timer1Timer()事件中，并且在它的事件处理中添加下列代码：

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
form1.ScrollBar1.Position:=form1.MediaPlayer1.Position
//显示视频播放进度
end;
```

由于控件属性设置的原因，在程序的执行过程中，每隔 100 毫秒的时间就会自动的激活一个 Timer1Timer()事件，通过语句

```
form1.ScrollBar1.Position:=form1.MediaPlayer1.Position
```

来随时的显示视频播放的进度。

5. 打开视频文件

在窗体上放置有一个“打开”按钮，它实现诸如打开对话框以及为设置播放文件等功能，所以在它的

Green Apple Data Center

Button8Click()事件中添加如下所示的代码：

```
procedure TForm1.Button8Click(Sender: TObject);
begin
if Form1.OpenDialog1.Execute then
    //显示对话框
    begin
    form1.MediaPlayer1.FileName:=form1.OpenDialog1.FileName;
    //设置播放文件
    form1.MediaPlayer1.Open;
    //打开多媒体播放设备
    form1.ScrollBar1.Min:=0;
    form1.ScrollBar1.Max:=form1.MediaPlayer1.Length;
    //初始化滚动条的状态
    form1.ScrollBar1.Enabled:=true;
    form1.Button1.Enabled:=true;
    end;
end;
```

在程序运行的过程中，用鼠标的左键单击“打开”按钮，就会自动的激活控件的 Button8Click()事件，程序首先调用 OpenDialog1 控件的 Execute 方法来显示一个提示用户选择视频文件的对话框，用户在该对话框中可以选择待播放的视频文件，之后程序就会通过语句 form1.MediaPlayer1.FileName:=form1.OpenDialog1.FileName;把用户的选择传递给多媒体播放设备，并且调用控件 MediaPlayer1 的 Open 方法来打开多媒体播放设备，然后通过一系列语句设置滚动条和其它按钮控件的参数和有效状态。

6．播放视频文件

在窗体上的按钮“播放”的作用是播放一个用户选中的视频文件，在 MediaPlayer 控件中有一个 Play 方法可以在程序运行的过程中播放一个打开的视频文件，为此，在按钮“播放”的响应过程中添加如下代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
form1.MediaPlayer1.Play;
//播放视频
form1.Button2.Enabled:=true;
form1.Button3.Enabled:=true;
form1.Button4.Enabled:=true;
form1.Button5.Enabled:=true;
form1.Button6.Enabled:=true;
form1.Button9.Enabled:=true;
//设置按钮的有效状态
end;
```

值得注意的是，在程序的运行过程中，只有用户在对话框中选择一个有效的视频文件，单击“打开”按钮，才能成功的向系统多媒体设备传送文件的路径和将要打开的文件名，这时单击“播放”按钮才能调用 MediaPlayer 控件中的 Play 方法来播放一个选中的音乐文件，并且设置窗体上其它控件的有效状态。

7．手动调节视频播放进度

在本示例程序中，对视频播放进度的调节是通过滚动条来实现，响应滚动条动态拖动的事件是 ScrollBar1Scroll，为此，添加如下所示的代码：

```
procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
   var ScrollPos: Integer);
begin
```

Green Apple Data Center

```
form1.MediaPlayer1.Pause;
//暂停播放
form1.MediaPlayer1.Position:=form1.ScrollBar1.Position;
//拖动到指定位置
form1.MediaPlayer1.Play;
//重新播放
end;
```

如果在程序运行的过程中用户用鼠标左键拖动滚动条上的滑动块，就会激活控件的 ScrollBar1Scroll 事件，程序首先调用 MediaPlayer1 控件的 Pause 方法来暂停视频的播放，然后把当前视频播放的进度通过语句 form1.MediaPlayer1.Position:=form1.ScrollBar1.Position;调节到当前滑动块的位置，最后调用 MediaPlayer1 控件的 Play 方法重新播放视频。

8．响应窗体的关闭事件

在进行多媒体程序设计的过程中，一定要注意这样的问题，就是在结束程序运行之前正常的关闭多媒体设备，虽然这样做对程序本身的运行不会带来太大的影响，但是养成一个良好的编程习惯对读者以后的工作肯定大有好处。

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
form1.MediaPlayer1.Close;
//关闭多媒体播放设备
end;
```

在程序结束运行时，首先会关闭窗体，通过代码 form1.MediaPlayer1.Close;来关闭多媒体播放设备，这样就不会发生多个应用程序之间抢占播放设备的问题了。

在这里对程序其它代码的添加过程就不一一加以介绍了，如果读者有什么不清楚之处，请参看附后的源程序代码。做完以上的工作后，存储文件。

9．运行程序

按键盘上的功能键 F9 运行程序。

在程序运行的过程中，由于没有给多媒体播放设备指定程序播放文件，所以窗体上的“播放”、“暂停”和“停止”等按钮都处于无效的状态。用鼠标的左键单击“打开”按钮，在屏幕上就会弹出一个提示用户选择视频文件的对话框。

在对话框中用键盘输入一个路径和文件名，或者用鼠标来选择一个有效的视频文件，单击“打开”按钮返回到程序运行的画面中，这时窗体上的“播放”按钮变为有效的状态，单击“播放”按钮，在窗体的 Panel1 控件上就会播放选中的视频文件，同时滚动条会动态的显示出视频播放的进度。

程序运行结果如图 9-20 所示。

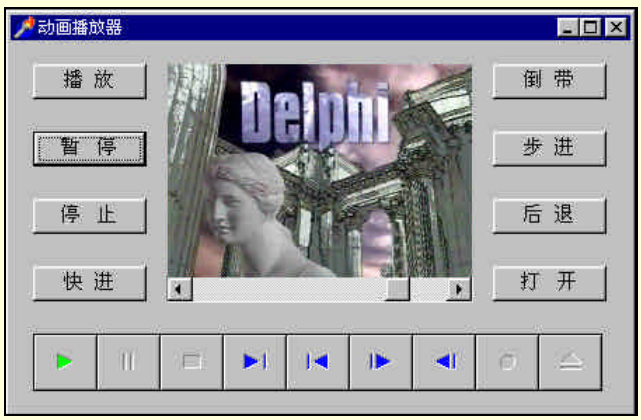


图 9-20 程序运行结果

在窗体上的各个按钮的功能如下：

■ “打开”按钮：打开一个对话框，从中可以选择一个有效的视频文件；

Green Apple Data Center

- “播放”按钮：如果用户打开了多媒体播放设备，那么单击这个按钮就可以播放指定的视频文件；
- “暂停”按钮：能够暂停视频文件的播放动作；
- “停止”按钮：停止播放视频文件；
- “快进”按钮：把当前的播放为止定位在视频文件的末尾；
- “倒带”按钮：把把当前的播放为止定位在视频文件的开头；
- “步进”按钮：根据控件的属性设置来控制单步前进的桢数；
- “后退”按钮：根据控件的属性设置来控制单步后退的桢数；

在上面的程序运行过程中，用户可以通过“暂停”、“快进”和“停止”等按钮对视频文件的播放过程加以控制，也可以用鼠标来拖动滚动条上的滑动块来改变视频播放进度。

完整的程序源代码如下所示，供读者参考学习。

程序清单

```
unit AVI;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls, MPlayer, ComCtrls;

type
  TForm1 = class(TForm)
    MediaPlayer1: TMediaPlayer;
    Button1: TButton;
    Panel1: TPanel;
    OpenDialog1: TOpenDialog;
    Timer1: TTimer;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    Button7: TButton;
    Button8: TButton;
    Button2: TButton;
    ScrollBar1: TScrollBar;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure Timer1Timer(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
```

Green Apple Data Center

```
    procedure Button2Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
    var ScrollPos: Integer);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
form1.MediaPlayer1.Visible:=True;
//设置控件的可见状态
form1.MediaPlayer1.Display:=form1.Panel1;
//设置播放容器
form1.Button1.Enabled:=false;
form1.Button2.Enabled:=false;
form1.Button3.Enabled:=false;
form1.Button4.Enabled:=false;
form1.Button5.Enabled:=false;
form1.Button6.Enabled:=false;
form1.Button9.Enabled:=false;
form1.Button8.Enabled:=true;
//设置按钮有效状态
form1.ScrollBar1.Enabled:=false;
form1.Timer1.Enabled:=false;
//设置控件有效状态
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
form1.MediaPlayer1.Play;
//播放视频
form1.Button2.Enabled:=true;
form1.Button3.Enabled:=true;
form1.Button4.Enabled:=true;
form1.Button5.Enabled:=true;
form1.Button6.Enabled:=true;
form1.Button9.Enabled:=true;
//设置按钮的有效状态
end;
```

Green Apple Data Center

```
procedure TForm1.FormClose(Sender: TObject; var Action: TCloseAction);
begin
form1.MediaPlayer1.Stop ;
form1.MediaPlayer1.Close;
//结束程序运行
end;

procedure TForm1.Timer1Timer(Sender: TObject);
begin
form1.ScrollBar1.Position:=form1.MediaPlayer1.Position
//显示视频播放进度
end;

procedure TForm1.Button8Click(Sender: TObject);
begin
if Form1.OpenDialog1.Execute then
    //显示对话框
    begin
    form1.MediaPlayer1.FileName:=form1.OpenDialog1.FileName;
    //设置播放文件
    form1.MediaPlayer1.Open;
    //打开多媒体播放设备
    form1.ScrollBar1.Min:=0;
    form1.ScrollBar1.Max:=form1.MediaPlayer1.Length;
    //初始化滚动条的状态
    form1.ScrollBar1.Enabled:=true;
    form1.Button1.Enabled:=true;
    end;
end;

procedure TForm1.Button7Click(Sender: TObject);
begin
form1.MediaPlayer1.Back;
//后退一格
end;

procedure TForm1.Button5Click(Sender: TObject);
begin
form1.MediaPlayer1.Previous;
//返回到文件开头
end;

procedure TForm1.Button4Click(Sender: TObject);
begin
form1.MediaPlayer1.Next;
//快进
end;
```

Green Apple Data Center

```
procedure TForm1.Button3Click(Sender: TObject);
begin
form1.MediaPlayer1.Stop;
//停止播放
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
form1.MediaPlayer1.Pause;
//暂停播放操作
end;

procedure TForm1.Button6Click(Sender: TObject);
begin
form1.MediaPlayer1.Step;
//步进式播放视频
end;

procedure TForm1.ScrollBar1Scroll(Sender: TObject; ScrollCode: TScrollCode;
   var ScrollPos: Integer);
begin
form1.MediaPlayer1.Pause;
//暂停播放
form1.MediaPlayer1.Position:=form1.ScrollBar1.Position;
//拖动到指定位置
form1.MediaPlayer1.Play;
//重新播放
end;
end.
```

9.4 小 结

本章集中的介绍了利用 Delphi 5 进行多媒体程序设计的一般思想，通过本章的学习，读者可以有重点的掌握 MediaPlayer 控件和 Animate 控件的应用，利用它们可以设计自己的“视频播放器”、“声音播放器”等多媒体应用程序。

在本章的最后，编者还希望读者能够在本章的基础上自己独立设计一个“媒体播放机”应用程序，在其中不但可以播放 Wave、Midi 等声音和音乐文件，还可以播放 CD 音乐等。

自此，本书全部讲解完毕。读者学习完本书的内容后，应该可以算是一个 Delphi 的熟练使用者了，不过要想成为 Delphi 编程高手，还需要一定的时日。所谓“十年磨一剑”，希望读者以后不断的加强编程练习，钻研更加高深的编程技术，早日成为编程领域的专家。

Green Apple Data Center